

SPIS: Technical Documentation		REF: SPISUI-TN01	
USER MANUAL - QUICK START		ISSUE 2.1	
		Date: 2010/10/25	
Abstract			
The present Technical Note (TN) presents the User Manual (UM), including a quick start, for the SPIS-UI framework. In the first part, this TN describes the User Interface (UI) and the various provided modules. In a second part, a complete simulation is followed step-by-step with a basic example. This document also introduce to the use of additional tools and functions, like the mesh inspection features, post-analysis tools, material properties settings and logging system. The main data structures handled by the common Data Bus and shared through the framework are also presented in a last part. This TN is compliant with the version 4.3 of SPIS.			
DOCUMENT STATUS SHEET			
ISSUE	REVISION	DATE	REASON OF CHANGE
0.1	JF	2004/11/20	Creation
0.2	JF		
0.3	JF	2005/01/07	Extensions
0.4	JF	2005/11/10	Update according to new GUI
0.5	JF	2006/06/10	Update
0.6	JF	2006/08/20	Update according to new GUI
0.7	JF	2006/09/01	Update
0.8	JF	2006/10/01	Extension/Correction
1.0	JF	2006/10/13	ISSUE 1, compliant with SPIS 3.6
1.1	JF	2007/04/13	Extension, correction, SPIS 3.7
1.2	JF	2008/02/20	Extension, correction SPIS 3.7 RC5
2.0	JF	2010/03/02	Extension, correction, SPIS 4.2 RC1
2.1	JF	2010/10/25	Extension, correction, SPIS 4.3
Responsible/Coordinator: Julien Forest (JF)			
Authors:			
Julien Forest (JF)			
Diffusion:	No restriction		

This page is left intentionally blank.

Table of contents

Introduction	5
Concept of Integrated Modelling Environment (IME).....	5
Modelling chain.....	5
Framework design	6
Multi-layer approach and data handling.....	7
SPIS-UI Quick Start	9
How-to-install SPIS ?.....	9
How-to-launch SPIS ?	9
GUI Overview	11
Project based approach.....	11
Creation of a new project	12
Open an existing project.....	12
Project structure.....	13
Pre-processing	16
CAD model edition and loading.....	17
Properties edition and loading	19
Material properties	23
Electric properties	24
Plasma properties	24
Groups settings and properties attribution.....	25
Meshing	26
Groups conversion and fields mapping	27
Global parameters settings	28
Additional parameters of the numerical core	29
Final conversion UI to Num.....	29
Numerical model generation and control of the simulation kernel.....	30
Multi-model approach and simulation process	30
Simulation daemon and local/remote simulation run.....	30
Dynamic edition and reloading of numerical kernels	32
Others SPIS-NUM parameters and settings	32
Post-processing and data analysis	34
3D Data analysis.....	34
DataFields Manager and grid conversion.....	34
3D data analysis and visualisation tools.....	35
2D data analysis tools and plotters	36
Other tools	37
Tasks and Data Bus management.....	37
Memory logger.....	38
Text editor and units converter.....	38
Message logger and information panel.....	39
Mesh Inspector and mesh quality control.....	40
Configuration and installation settings	43
Management and compilation tools	44
Templates, examples projects and tests	45
Online help and documentation.....	45
Step-by-Step how-to-drive a modelling process?.....	47

Advanced use.....	49
Data structure, DataFields and MeshFields.....	49
Handling objects and data from the console.....	51
Jython completion	51
Task building and execution.....	52
Manual configuration, execution and manipulation of the SPIS-NUM kernel	53
SPIS-UI in batch mode and use of SPIS's tracks	54
Supported platforms	54
Java and native layers : handling of the full or degraded modes	54
Memory cost and limitations	55
Difference between 32 and 64 bits systems ?.....	56
References	57
Annexe 1: Launching scripts	59
Unix like configuration scripts	59
Windows.....	61
Annexe 2: examples of SPIS-UI tracks	63
Processing script.....	63
Simulation demon.....	64
Annexe 3: JyTop4.py script.....	66
Annexe 4: Task dependence tree, tasklist.py script.....	76
Annexe 5: Example of NASCAP material catalog.....	80
Annexe 6: Example of extended material	84

1.Introduction

1.1. Concept of Integrated Modelling Environment (IME)

SPIS-UI is the user interface of the SPIS system. The functions of SPIS-UI include the possibility of definition of all inputs parameters (CAD geometry, material properties, numerical settings...), during the pre-processing phase, the control of each step along the whole modelling chain (Fig.1) and the processing of outputs after the simulation. In parallel, the SPIS-UI framework offers the possibility to access to all shared data and dynamically edit and reload all models and processing modules.

1.2. Modelling chain

One of the main objectives of SPIS-UI framework is to help the user to follow the modelling chain and perform each step in a consistent approach in order to guaranty the results of the simulation as possible. The most common process starts with the definition of the geometry of the spacecraft (GEOM) with Computer Assisted Design (CAD) module and the attribution of surfaces properties and boundary conditions. This phase corresponds to the *pre-processing phase*. At its end, the system to model should be fully defined and ready to be loaded into the simulation kernel, SPIS-NUM, for the *Simulation Phase*. SPIS-UI provides several ways to run the simulation kernel, inside the framework, in interactive mode, or as independent jobs (under UNIX based systems) for large simulation or execution on a remote computer. The modelling process is closed by the post-processing phase with the access to several data analysis modules and 2D/3D visualisation tools. Fig.1 illustrates this principle.

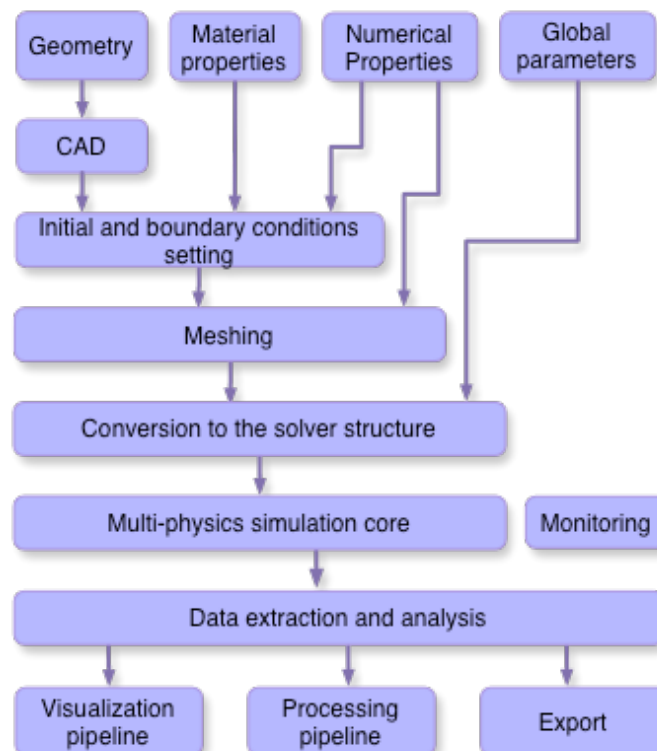


Figure 1: Schematic modelling chain.

1.3. Framework design

SPIS-UI is a modular framework building the functional link between all tools used along the modelling chain, from the pre-processing phase to the post-processing phase and including the simulation phase.

SPIS-UI is designed around a central kernel constituted of a **Common Data Bus**, to help the data exchange between components of the framework, and a central workflow, called **TaskManager**. Each functional module, like CAD tool, mesher, simulation kernel or 3D viewers, are embedded into a generic container call **Task**, and dynamically loaded as a plug-in into the framework. Fig.2 shows the SPIS-UI structure.

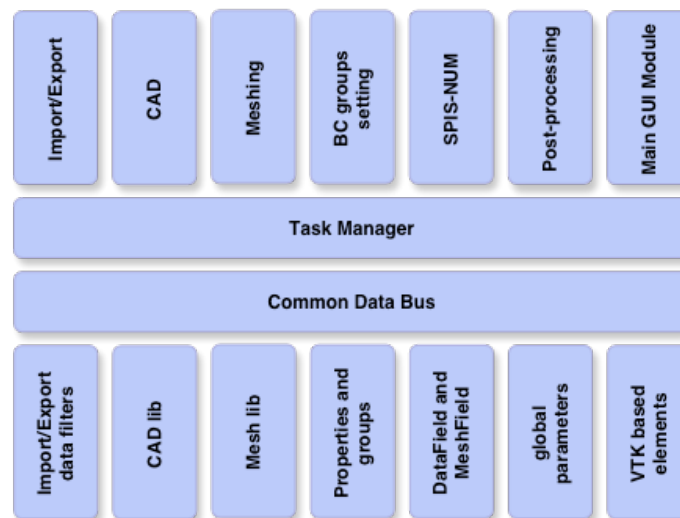


Figure 2: Structure of the SPIS-UI modelling framework

Technically, the SPIS-UI framework is written in Python/Jython script language and wraps sub-components in Java or native languages through a JNI interface. The Graphic User Interface (GUI) is fully written in Java Swing. Fig.3 illustrates this approach with several types of languages.

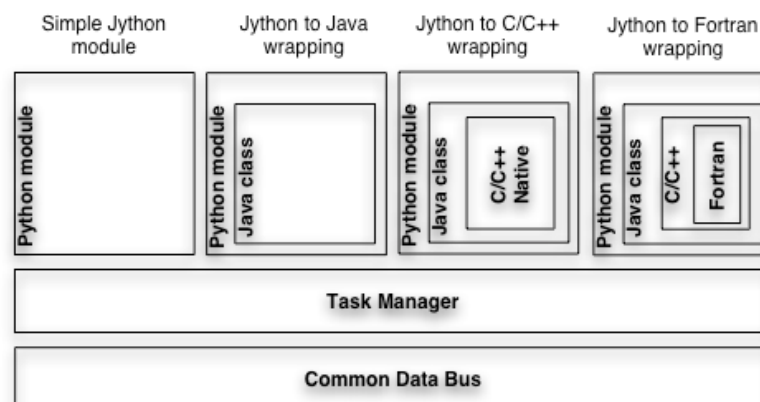


Figure 3: Principle of multi-language integration

This choice leads to a lightweight solution and allows a very high portability of the whole system. The SPIS-UI framework it-self can be run on almost all platforms with a Java Virtual Machine (JVM) 1.4 and higher. For each native component, a binary image is provided the

most common OS (Windows, Linux, Mac-OSX...). The native modules are dynamically loaded into the framework through a mechanism of control (exception), allowing to still run the framework in degraded mode if one of them cannot be loaded. More information is available on the collaborative platform of the SPINE community [2].

The purpose of the *Task Manager* is to help the user to follow a correct order each step of the modelling according to a dependency tree between each task or pre-defined scenarios under the form Python scripts. The Task Manager works in a manner similar to a simplified workflow or a Make script. Fig.5 below illustrates this principle.

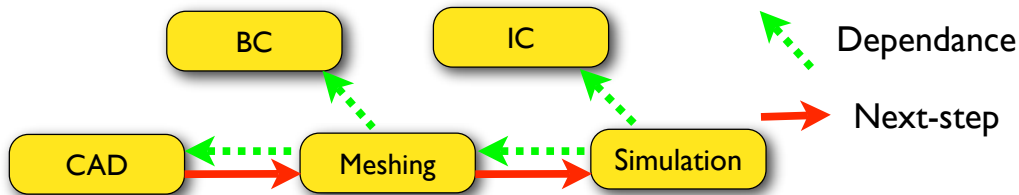


Figure 4: Illustration of the principle of the TaskManager: If a task (i.e module calling) is dependent on other ones (green arrows), these tasks will be automatically performed before in a correct order in order to generate consistent data. Scenarios, called SPIS tablatures, can be defined to perform each task auto

1.4. Multi-layer approach and data handling

The SPIS-UI framework has been designed to offer several levels of User Interfaces (UI) to access to tools and manipulated data:

- **A Graphical User Interface (GUI):** A rich GUI offer the possibility to run all functionalities of the framework through a set of menus, control widgets and graphical editors. This approach is especially useful for pre-built processes, as during a classic study. The GUI of SPIS-UI can be easily extended and adapted through an XML descriptor, without any compilation. Please see the TN 4.0 Developer Guide for further information [4].

- **A Script Language (SL):** In order to offer a prototyping capability and to facilitate the data handling, Jython, a Java based Python interpreter, has been integrated into the framework [1] [8]. Most of the the top level modules of SPIS-UI are written in Python/Jython and can be easily modified and manipulated. A plug-in based mechanism allows to dynamically editing and reloading most of the components of SPIS into the framework without any recompilation or necessity to re-start the framework. This feature is especially useful to prototype and dynamically test models on a set of data already loaded in the framework. The script capabilities are completed with the introduction of JyConsole, an advanced Jython console, providing an Object Oriented Completion Function (OOCF).

- **A Source Code Direct Access (SCDA):** As well as for the script layer, most of the low-level Java libraries (e.g SPIS-NUM) can be edited, re-compiled and dynamically re-loaded into the framework.

Some modules, like the 3D viewer Cassandra [3], and most of the external tools can individually be run as standalone applications or launched as independent demon.

This page is left intentionally blank.

2. SPIS-UI Quick Start

2.1. How-to-install SPIS ?

In default mode, SPIS does not require any specific installation. A release tar ball is generally available for most of the common platforms. Each to them provides all needed elements for the targeted platform, including external libraries, tools and JVMs in the `$SPIS_ROOT/ThirdPart` directory.

The SPIS releases are available on the SPINE's collaborative platform at the following address: <http://www.spis.org> [2].

The releases are archived as tar and gzipped or zipped files. For Linux, Mac OS-X and Windows systems, please just untar the release into your chosen working directory. For other platforms, a JVM and native components should be installed specifically. Please see the SPINE platform for further information [2].

The untared image can be also copied on a CD-ROM and directly run from it.

Remark: In some cases, if another JVM was installed before on your computer, some environment variables can be set and interfere with the launching scripts of SPIS. Please see the SPINE platform for further information.

Supported platforms and advanced installation settings are discussed in [section 4.3 Supported platforms](#).

2.2. How-to-launch SPIS ?

SPIS is a Java based application called through a Jython script. SPIS can be run on almost all platforms with a JVM. Only native modules, such as VTK or the meshing tool, are platform dependent. If theses elements are not properly set, SPIS generally starts in degraded mode with only the Java based modules loaded.

Several launchers have been developed to facilitate the launching of SPIS on various platforms. Depending on the version of SPIS, there are available directly under the root directory of `$SPIS_ROOT` or in the `$SPIS_ROOT/SPIS-UI/BinXXX` directory, where XXX corresponds to the relevant platform. The table here below summarises the launching scripts/ platform configurations:

System	Launching script
Linux	runSpis.sh
Apple MAC OS-X	runSpis.sh
MS Windows NT/2000/XP	runSpis.bat

Excepted for the Windows case, these scripts must be run in a shell (terminal). Basically, theses scripts define all required environment variables and paths toward sub-components (libraries and tools). The full SPIS releases are generally provides with all needed external components, such as CAD/mesher tools, viewer, editors, including JVMs for most common platforms. Please browse the `$SPIS_ROOT/ThirdPart` directory. By default, they are set with respect to components provided in the `$SPIS_ROOT/ThirdPart` directory for further

details. They generally do not need any modification or specific settings. However, on some platforms or if you wish to optimise the global performances, the environment variables defined inside may have to be corrected manually. The launching scripts are largely commented and given in Annexe 1.

In complement of the SPIS application itself, several of its components can be run as standalone applications. The table here after list these tools and the short description of their function:

Script name	Tool	Description
Cassandra.sh	The Cassandra 3D data viewer. See http://www.artenum.com/FR/Produits-Cassandra.html for further informations.	Post-processing module to perform 3D data analysis and visualization. Also used to visualize geometrical elements and groups setting in the pre-processing phase
JyConsole.sh	JyConsole, see http://dev.artenum.com/projects/jyconsole for further information.	A Jython console, with an object and syntactic completion function.
JySpreadSheet.sh	JySpreadSheet.	A generic units converter.
runGeomManager.sh	The SPIS/Geom Manager as standalone application.	STandalone version of the geometry manager.
runMeshInspector.sh	JFreeMeshInspector	The JFreeMeshInspector adapted to the SPIS contexte.
duplicateProject.sh	Shell script	Duplicate all input to a given project into a new one.
runPropertyEditor.sh	Artenum Property Editor	Graphical properties editor to adjust the SPIS settings (if needed).
spis-ui.properties	General configuration file.	This file set most of all paths and environment settings needed by all SPIS applications. It is called by most of all other launching script.

Remark: Under Windows NT/2000, the DOS prompt command does not support environment variables longer than 256 characters and does not support the launching script. This problem can be solved by running the XP version of the prompt in place of the standard one or by running the Sygwin environment. Under MAC OS-X the VTK module is still experimental. A specific installation of the VTK library is needed. Please see the Cassandra's Web site for further information [3].

2.3. GUI Overview

The SPIS user interface is based on a Main Graphical User Interface (MGUI). The MGUI looks like an integrated desk, where all individual GUIs of each sub-module are projected. The MGUI provides a central menu and a tools bar to control and call all modules and components of the framework. On the tools bar, the order of buttons corresponds to the typical order to perform the tasks in a standard modelling process. Two internal sub-frames complete the GUI. The first frame is the logs window where all messages, errors and outputs are reported. This frame includes actually two different logs windows for each language layer (script and Java based modules). The second one is a Python/Jython console (i.e JyConsole), where you can write and execute all types of Python scripts and manipulate directly all object shared in the framework. JyConsole offers an object-oriented completion for both JAVA and Jython languages.

The main window is defined dynamically through a simple XML based descriptor and can be very easily extended and adapted without any JAVA coding or re-compilation. Please see the advanced section or the TN 4.0 SPIS-UI Developer Guide for more details [4].

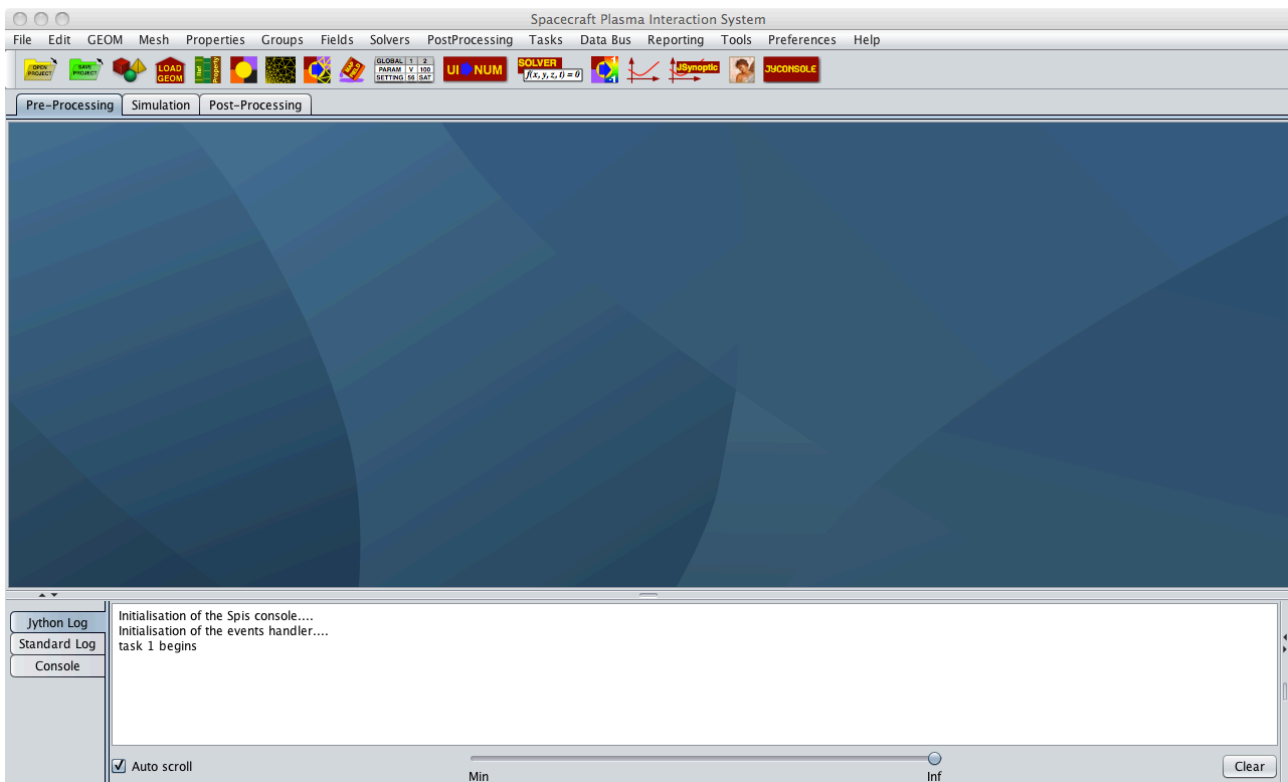


Figure 5: General view of the main GUI of SPIS.

2.4. Project based approach

The persistence model of the SPIS-UI modelling chain stands on a project based approach, where all needed parameters and actions performed during each step are progressively saved. Before any new study, we recommend to define a new project first. This recommendation is especially true before performing large and complex simulations. A SPIS project links all needed data (CAD file, mesh, groups settings, material, global parameters...) needed to perform the simulation and recover computed data. Scalar and vector fields

applied on the mesh (DataFields) are also saved and can be reloaded for visualisation and post-processing. VTK files produced in output of the post-processing modules (Cassandra, Paraview) can be also saved into a SPIS project.

Practically, a SPIS project is a directory with a set of pre-built subdirectories and files. It can be packaged with classic archives tools, like tar or zip, to be moved to another place or computer. Most of the data are saved under the form a Python/Jython scripts, obtained by serialisation of the corresponding SPIS objects. These scripts are in ASCII format and can be directly integrated in all Jython based programs with the classic Python `import` command.

The project structure is still very modular and each component can be loaded independently and new projects built by aggregation of existing components. The current project format is not stabilised and can be modified without any notification.

2.4.1 Creation of a new project

To create a new project, please choose the menu **File -> Save Project** or push the  button in the tool bar. A first directory browser should appear. You can directly write the name of your project directory or use the browser to choose a pre-existing directory. The project creation phase will build all needed sub-directories and files.

The project will progressively built and saved, element by element, along the modelling chain every time you click on the  button or on the **File->Save Project** menu.

2.4.2 Open an existing project

A project can be loaded at different levels, depending on the data you wish reload and the part of the pre-processing phase you want proceed again. First, click on the  icon or on the **File -> Load Project** menu. As illustrated in Fig.6, a dialogue window will appear.

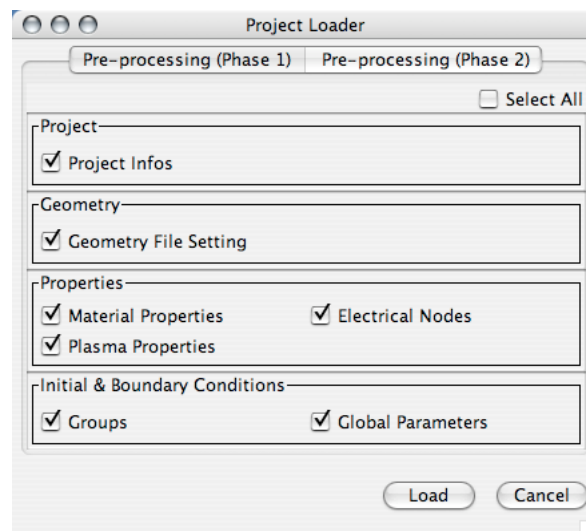


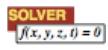
Figure 6: Dialogue window of the project-loading phase. Most of the project's components can be loaded individually.

The first index of the loading windows set whole minimum set of elements needed to define a complete system. They are enough to start the modelling process, i.e set the reference to

the CAD file, the material and plasma properties, the group attribution and all global parameters. At the end of this phase the system is ready to be pre-processed, i.e meshed and local fields deployed.

Most of component can also be loaded individually. This approach can be used to recover one component only, like a property for example, or parts of a corrupted project. The second page corresponds to the elements produced during the second step of the pre-processing phase, such as the mesh structure and the local fields (i.e DataFields and MeshFields). This step is required is you wish recover results of a previous simulation (i.e DataFields). For large systems, this second step of the loading phase may be long. Please open the logs window and check if the loading phase is completed before to continue the modelling process.

At the end of the loading and if the system is fully defined, the framework is ready for a new simulation. Thanks to the TaskManager, you only have to push the **SpisToNum** button to perform automatically of the whole pre-processing phase. Only required tasks will be done. If the first loading step was done only, the TasksManager will call automatically the mesher and all tools needed to deploy the local properties (IBC). If the mesh and pre-processing DataFields are already reloaded (second part the loading phase), the TaskManager will only call the UI to Num data structure converter.

At this level, SPIS-NUM, the numerical kernel can be called by pushing the  icon in the tools bar.

Remark: The project loading procedure can be modified in the future. Please see the SPINE Web site (<http://www.spis.org>) for the possible evolutions.

2.4.3 Project structure

All data of a SPIS-UI project are gathered into a root directory and a set of sub-directories, as illustrated in Fig.8 below. The structure is design to be the most modular as possible and allow future extensions. Future versions may have additional sub-directories and data. Under the condition to respect the consistency to the stored model, each data is an independent entity. During the loading phase, the loading module of SPIS-UI will try to load each data individually. If one data is corrupted, this does not stop the loading of the other ones. However, the user must be careful. In this case the project may be incomplete and inconsistent to run a simulation.

In the current format (3.0), most of the SPIS-UI native data are saved under the form of ASCII Python scripts. They can be loaded independently in all Jython interpreter or console, like JyConsole. To change their settings, you can directly edit them with a standard text editor, without use the GUI of SPIS. This approach is useful in batch mode or to change the settings of a project on a remote computer.

A SPIS project is structured as follow:

- **__init__.py file**: This empty file is only present to indicate to the Jython interpreter this directory and the Python modules contained in it should be considered as a Python module.
- **DataField directory**: This directory contains all deployed local fields (i.e DataFields) and their corresponding set of mesh elements (i.e MeshFields). Each

DataField is stored under the form of a set of Python scripts gathered in a dedicated sub-directory.

Remark: DataFields and MeshFields require a pre-existing mesh structure to be deployed. You must then imperatively load the mesh into the framework before to load the DataFields. If the fields have been generated during a previous simulation, the mesh structured must also be exactly the same than during the initial construction. For a same GEOM structure, we notice that regenerated meshes , obtained by calling the mesher may differ, with as a consequences the impossibility to re-load the DataFields. We observe that the result of the mesher is sensitive on the OS (effect of round error) and its settings (conversion criterion).

- **Geom directory:** This directory contains all GEOM files describing the geometry of the system (spacecraft and computational domain). A Python based description file completes the settings of the GEOM workspace, setting the main GEOM file, for example. The contains of this directory is the image of the workspace of the GEOM manager.
- **Groups directory:** This directory contains all scripts setting and describing all geometrical groups.
- **Images directory:** This directory contains all images produced by the post-processing modules (Cassandra, 2D plot viewer).
- **NumKernel directory:** Because implemented the numerical kernel (SPIS-NUM) has specifics and additional parameters, like sources characteristics or the internal electrical , depending on the built model and not managed by SPIS-UI, an additional sub-directory has been added to the project structure.
- **Properties directory:** This directory contains all properties being deployed on the structure through the group description, like material properties or initial and boundary conditions. By default, three sub-directories correspond to the Plasma properties (i.e initial and boundary conditions), Material properties and Electrical nodes description.
- **Reporting directory:** This directory is dedicated to centralise notes and reports in future evolutions of SPIS-UI.
- **Log files:** These ASCII files contain all the outputs of the various consoles.
- **spis-global file:** This file contains the setting of all global parameters. Currently this file is obtained by serialisation. In future versions of SPIS-UI this file will be saved as Python script.
- **spis-names files:** This file is a reference and control file for the project. Please do not edit it manually.
- **spisSimulationDeamon files:** These files are scripts files to perform the simulation steps in batch mode. Please see the corresponding section for further information.

- **Tmp3D.msh file** is the default mesh file (Gmsh format [6]) generated or loaded by the framework.
- **VTK directory:** This directory contains all VTK data obtained after conversion from UI. These files can be loaded and processed independently on SPIS-UI.

Remark: The structure of SPIS-UI project can change without notification. Please see the SPINE's community Web site for further details and migration procedures.

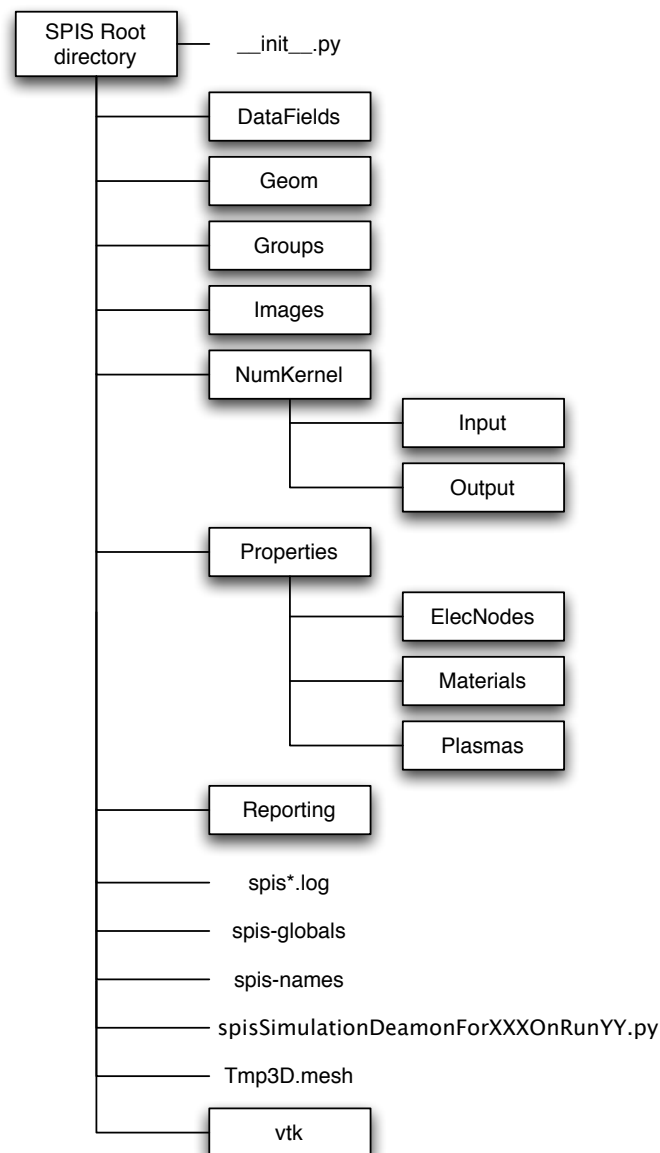


Figure 7: Structure of a SPIS project. Most of the components can be loaded individually and are Python based scripts.

2.5. Pre-processing

The pre-processing phase corresponds to the construction phase of the modelled system. This is the phase where all parameters and settings needed by the simulation kernel are set. This includes the GEOM model, the local parameters, with a different value depending on the localisation on the mesh like IBC, and global parameters, with a value common to the whole system, like the time step. Fig.9 below summarizes the pre-processing process.

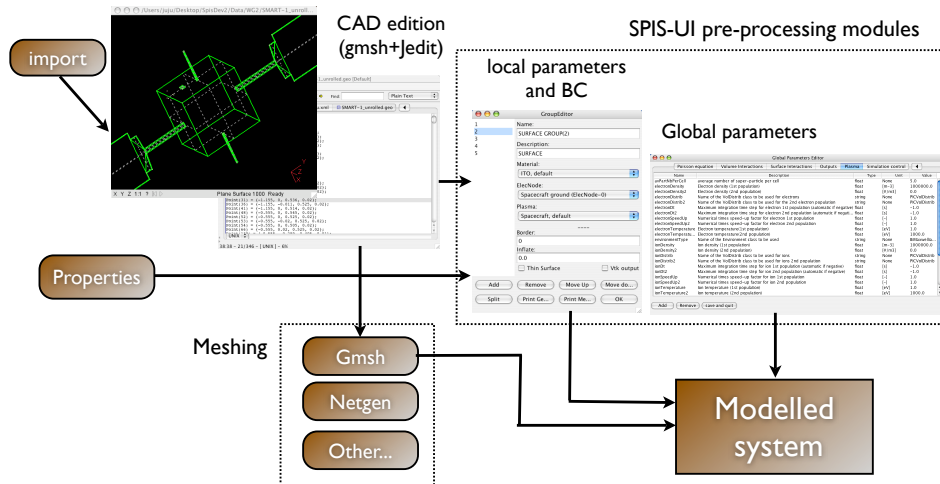


Figure 8: Synoptic scheme of the pre-processing phase.

One of the key setting is the link between the geometry of the system (spacecraft and computing space) and the initial and boundary conditions. In the SPIS-UI framework, these parameters are generally called “local parameters”, in reason their value may differ on their localisation. These global parameters are Properties associated to the GEOM model through a set of geometrical zones, called “physical” in the Gmsh format or GEOM groups in SPIS-UI. SPIS-UI provides a set of tools to attributes properties to groups (i.e Group Editor) and to deploy these properties on the mesh (i.e MapFields functions and Fields Manager), as illustrated Fig.9.

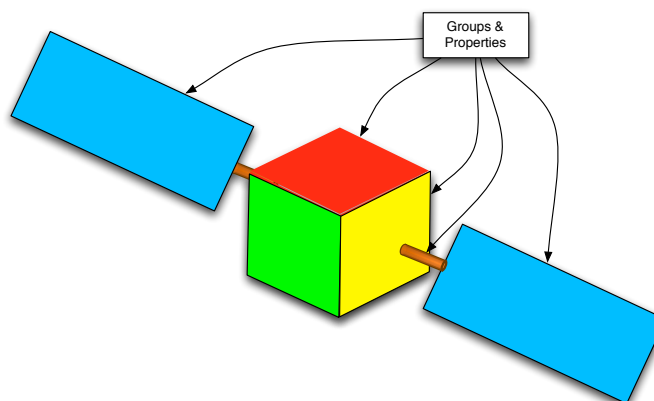


Figure 9: Example of groups and properties setting. The spacecraft will be sub-divided into several “Gmsh Physicals” to which ones, the user can associate properties, using the Group Manager of SPIS-UI. In a second step, these properties are converted into local fields (DataFields) deployed on the mesh and passed to the simulation kernel SPIS-NUN as Initial and Boundary Conditions (IBC).

CAD model, groups edition and settings of local properties is a very delicate and complex phase. This is generally the main source of mistakes or failures in the modelling process. Tasks must be done step after step and settings consistent with the mathematical model of the solver. The internal **TaskManager** of SPIS-UI must help the user to perform **Tasks** in the right order. If a **Task** is called in downstream all (*) needed **Tasks** in upstream will be called automatically by the **TaskManager**.

(*) Remark: This is done if all Tasks are linked by an explicit dependence tree and can be automatised. Some **Tasks**, like as the **Group Manager** cannot be done automatically.

2.5.1. CAD model edition and loading

SPIS-UI supports complex geometrical descriptions with a multi CAD files description. To help the CAD files edition and the management of geometry and CAD file, SPIS-UI provides a GEOM Manager, which gathers into a common workspace all relevant CAD files and associated document. Currently SPIS-UI import Gmsh file format only. By clicking on the  icon, a control frame must appear as illustrated in Fig.10.

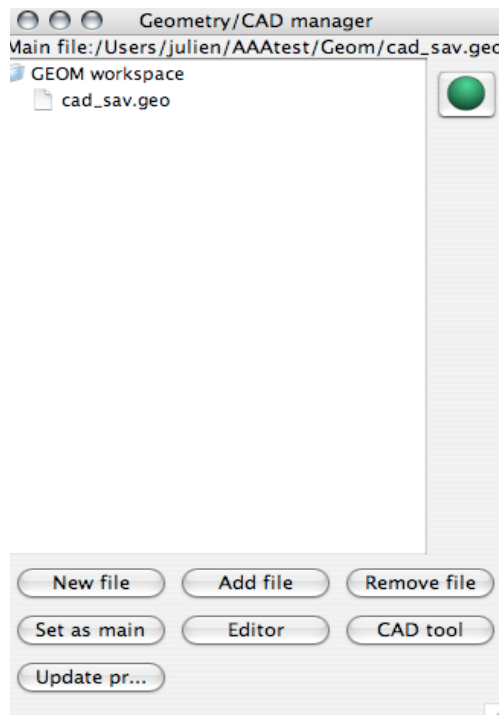


Figure 10: View of the GEOM Manager.

The purpose of the GEOM Manager is to help the user to edit, modify and organise all relevant CAD files of this current project. The **New File** button will create a new empty file. It is then possible to select it in the tree and edit it either with the default CAD tool, by clicking on the **CAD Tool** button, or with the default ASCII editor, by clicking on the **Editor** button. It is also possible to add an existing CAD file to the workspace by clicking on the **Add File** button.

Important: In the same logic than with Gmsh, the whole geometry can be defined through several CAD files using the Gmsh “include” command in a main file (please see the Gmsh

Technical Documentation for further information). However, SPIS-UI will import only this main file and unrolled the whole structure. For this, you must absolutely defined this file as "main file" for the framework, by selecting the relevant file in the tree and clicking on the **Set As Main** button. This operation must be absolutely done before the loading procedure.

The **Geom Manager** provides also a set of pre-built CAD components. For example, to click on the icon  will open a GUI for the generation of a pre-built sphere. This sphere can be translated and scaled. This tool does not provide any Boolean operations between generated objects. However, it is possible to export ether nodes or wires only or the whole CAD structure.

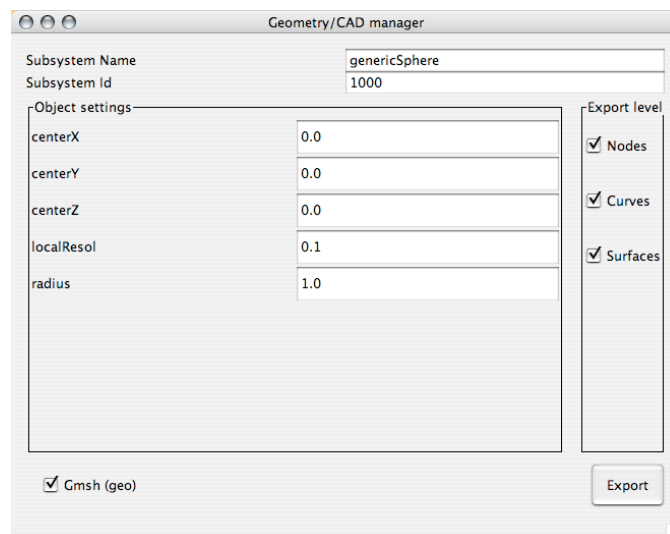


Figure 11: View of the pre-built CAD objects editor (here a pre-built sphere).

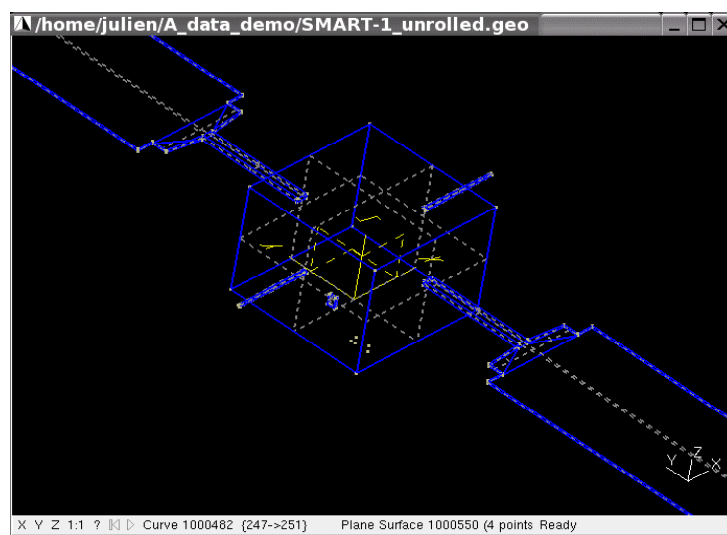


Figure 12: Example of CAD geometry, model of the ESA/SMART-1 mission

As introduced above, we outline the fact that several "**Physicals**", in the Gmsh sense, must be defined to set all Boundary Conditions (BC) needed by the solver. This generally

depends on the system to model and the constraints of the numerical kernel. However, at least, the three following **Physicals** must be defined.

<i>Object or zone to be marked</i>	<i>Type of physicals</i>
Spacecraft surface	Surface
External boundary of the computational volume	Surface
Computational volume	Volume

For more details regarding the CAD edition, please read the "Gmsh User Tutorial for CAD and Meshing" and the SPIS-NUM Documentation "How to control Num from UI" [6][13].

The edition of the various CAD file will only define the geometry of the system in the workspace of the GEOM manager. At this stage, the CAD structure is not loaded into the framework. At this stage a **Main GEOM File** must be set, with the **Set As Main** button, the corresponding CAD structure must be loaded into the framework by clicking on the  button.

NB: The GEOM Manager edits and modifies file in its own workspace in the temporary exchange folder of SPIS and not directly files in the project. For this reason, we recommend to update your project regularly by clicking on the **Update Project** button.

2.5.2 Properties edition and loading

In SPIS-UI, the notion of properties corresponds to definition of local parameters, like boundary conditions, used in the definition of the model. This includes material, physical and numerical properties. As introduced above, the purpose of the pre-processing part is to attribute these properties to each specific parts of the spacecraft or the computational domain, convert them into fields (i.e DataFields) mapped on the mesh.

Properties are divided into four major categories:

Property type	Purpose
Numerical Material Properties	Define the numerical flags and parameters needed to define material models conditions corresponding to physical surfaces, like the S/C surface. These properties may refer to built-in material or externally defined material, like NASCAP based, through the related properties.
Electrical Node Properties	Define the numerical and physical properties related to the electrical node models.
Plasma Model Properties	Define the numerical properties and boundary conditions related to the numerical plasma models. Various models are defined, in function that this property should correspond to a material boundary (i.e S/C surface), a virtual boundary (e.g external boundary of the computational domain) or the whole computational volume.

Property type	Purpose
Related Properties	Define related properties that can be connected to the previous one. This can be used to add data and/or other parameter to a given property.

Each of them are described in more detail here after.

A large catalog of preset properties is defined in the `$SPIS_ROOT/SpisUI/Modules/Properties` directory and can be loaded by clicking on the  icon in the tools bar or through the **Properties->Load Default Properties Catalogue** menu. It is also possible to load individually external properties through the **Properties->Load Default Property** menu. A selection panel as follow will appear.

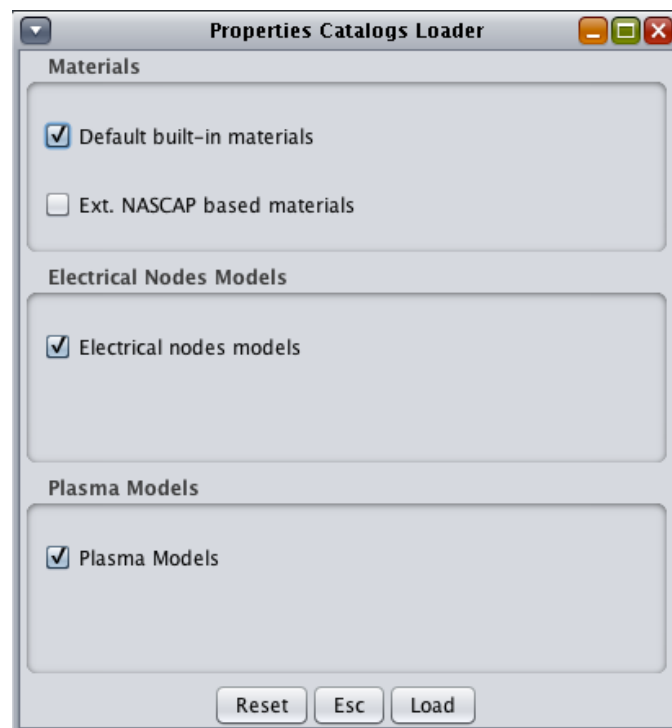


Figure: Selection panel of the properties loader

Each type of property (Material, ElecNode and Plasma) have the same logical structure summarised as in the figure below.

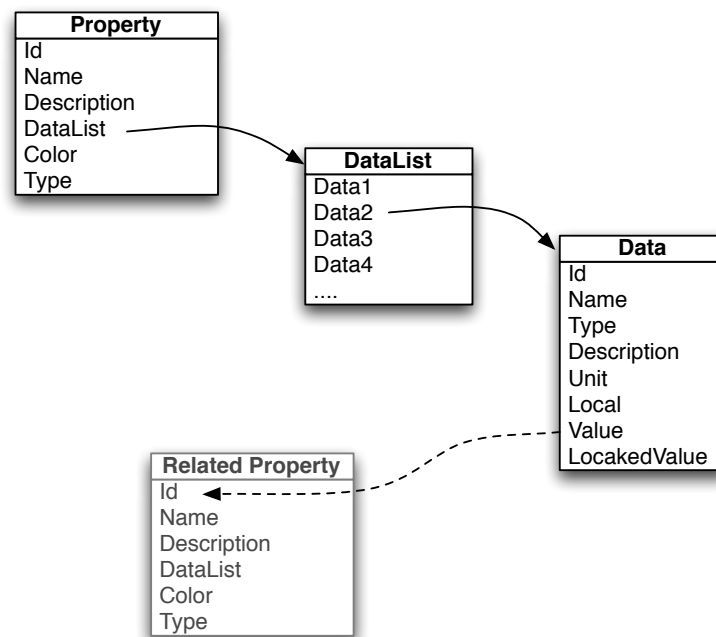


Figure: Logical structure of the all properties.

Properties store directly several meta data (e.g Id, name, description) and through a DataList object the physical and numerical information (i.e Data). For each Data object, a set of values can be defined, with its unit, dimension and localisation of the mesh cells.

Properties can be chained through dedicated in their DataList, in order to extend their capabilities, like for NASCAP based materials, for instance.

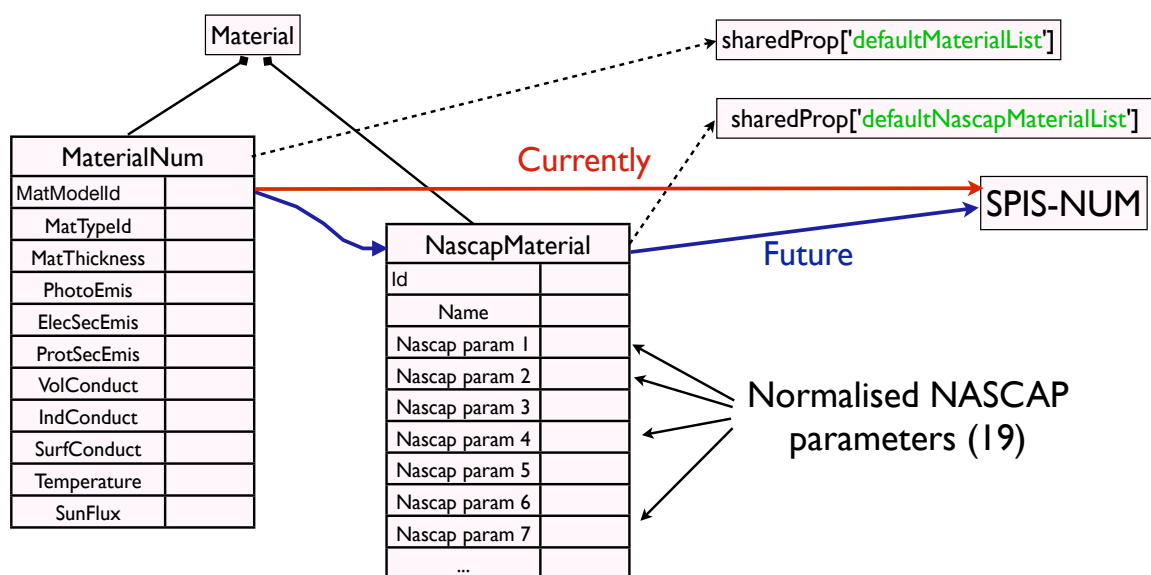
Most of properties are stored in RAM Data Bus in the sharedProp["defaultMaterialList"] dictionary.

In addition, catalogs of NASCAP based material property can be imported from external XML based file (NASCAP 2K format). For each defined material, this will automatically generates a default numerical material, with all fields and flags required by SPIS-NUM, and the related property, here defined as NASCAP material, with the physical characteristic of the material. The physical characteristics of the material are defined according the 19 canonical NASCAP parameters reminded here below.

0: ITOC (Material coated with ITO)
 1: CERS (Solar cell material. Cerium doped silicon with MgF2 coating)
 2: CFRP (Carbon fibre, conducting, no resin layer)
 3: KAPT (Kapton, average values for SEE...)
 4: COSR (Optical solar reflector without MgF2 coating. Cerium doped glass type)
 5: EPOX (Epoxy. Thin layer of Epoxy resin on (conducting) Carbon fibre)
 6: BLKP (Non conductive black paint. SEE yields are as measured for Electrodag 501)
 7: BLKH (Non conductive black paint HERBERTS 1002-E. Values updated 3.10.88.)
 8: BLKC (Conductive black paint Electrodag 501)
 9: PCBZ (White paint PCB-Z assumed to be conductive in space)
 10: PSGI (White paint PSG 120 FD assumed to be conductive in space.)
 11: TEFL (Teflon, DERTS measurements of SEE)
 12: CONT (Generic Dielectric after 5 years in GEO environment.)
 13: GOLD
 14: SILV (Silver as from NASCAP library)
 15: ALOX (Oxydized Aluminium. SEE yields from DERTS for Aluminium/Kapton)
 16: STEE (Steel, SEE sigma +E_{max} from DERTS, curve shape from CONT material)
 17: AL2K (Aluminium according to NASCAP-2k)
 18: AU2K (Gold according to NASACP-2k)
 19: KA2K (Kapton according to NASACP-2k)
 20: TE2K (Teflon according to NASACP-2k)
 21: OSR2K (OSR according to NASACP-2k)
 22: BK2K (Black Kapton according to NASACP-2k)
 23: SC2K (Solar Cells according to NASACP-2k)
 24: NP2K (Non-conductive paint according to NASACP-2k)
 25: GP2K (Graphite according to NASACP-2k)

List of the canonical NASCAP parameters defining a material.

The figure here below shows, as example, how complex NASCAP based materials are defined.



Memory structure of NASCAP based materials.

NASCAP based material catalogs can be imported through the Properties

Properties can be edited, modified, duplicated or suppressed, through the SPIS-UI interface and the Properties Editors, be saved and by the way extends the available set of properties.

2.5.2 1 Material properties

Material properties define the physical characteristics properties of the material. This includes basic properties like as conductivity or secondary emission factors, but also advanced definition like as the characteristics of a source of particles. Typically these properties are attributed to the spacecraft only.

They can be edited and extended using the **Material Editor** (menu **Properties -> Edit Properties -> Material Editor**).

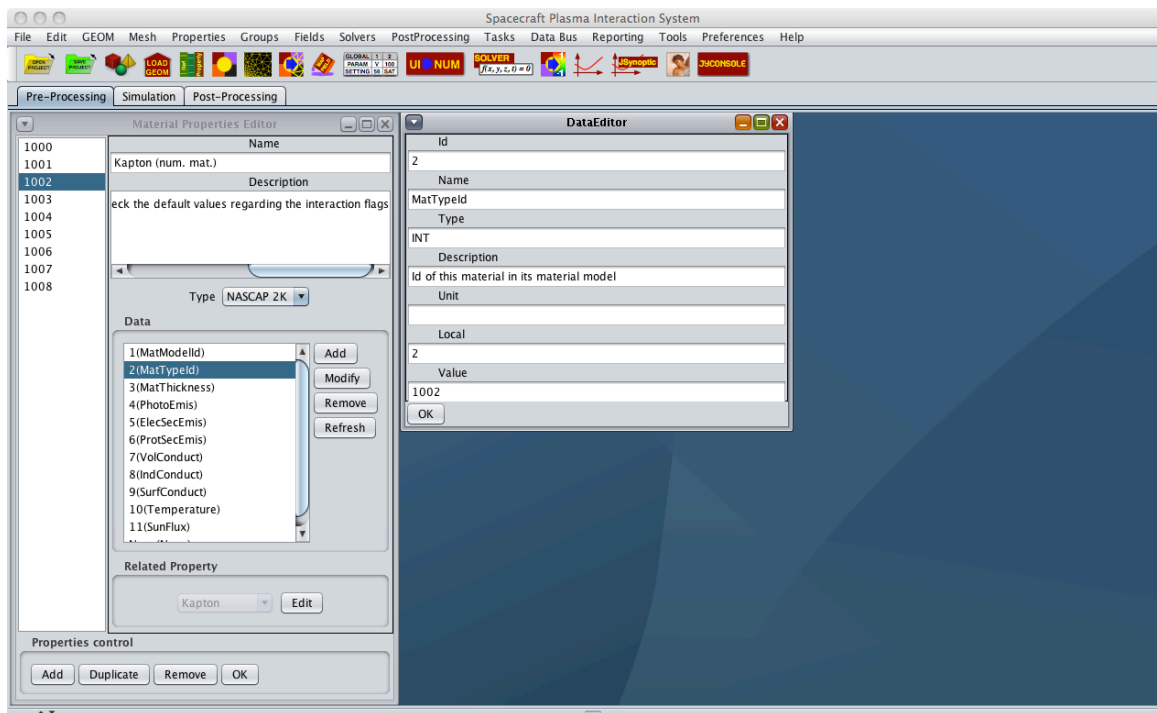
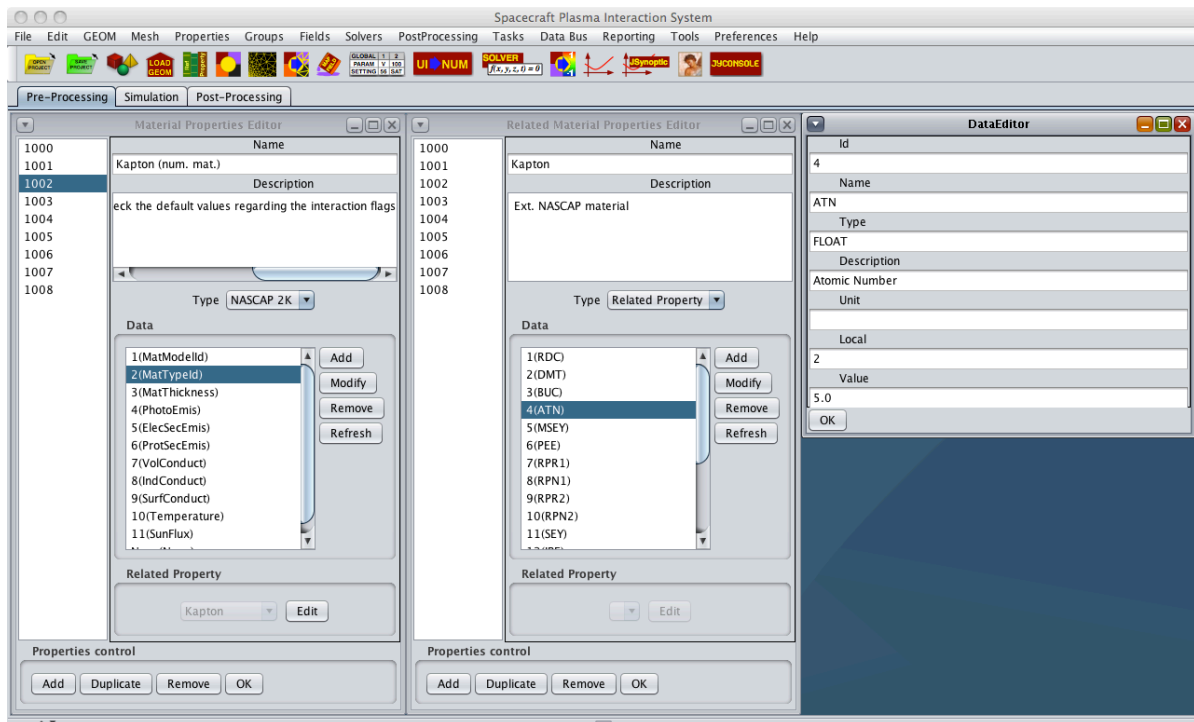


Figure 13: Properties Editor.

The related property can be edited in the same way, just by clicking on the **Edit** button in the Related Property section.



Edition of related properties.

2.5.2 2 Electric properties

The electric properties are related to the definition of the internal charging balance and, more specifically the attribution of main electric node in the spacecraft model. There setting can be done using the **ElecNode Editor** (menu **Properties -> Edit Properties -> ElecNode Editor**). Typically these properties are attributed to the spacecraft only.

2.5.2 3 Plasma properties

The **Plasma Properties** correspond to the numerical setting of boundary and initial condition of the plasma numerical model. Its structure and use is strongly dependent on the used model and its numerical constraints. A wrong definition or attribution of the **Plasma Properties** can lead to forbid to run the simulation kernel or to lead to a corrupted simulation. Please read carefully the SPIS-NUM documentation for more details [13]. These properties must be defined for the whole computational domain. They can be edited and extended using the **Material Editor** (menu **Properties -> Edit Properties -> Plasma Editor**). Because they are strongly dependent on the structure of the simulation kernel, we strongly recommend to do not edit them without a deep knowledge of the SPIS-NUM design and models.

All properties can be edited and corrected using the Properties Editor, which can be called with the **Properties->Edit Properties** menu. Fig.13 show an example of property edition.

2.5.3 Groups settings and properties attribution

In order to define the properly the needed IBC, the various properties should be attributed to there respective Physicals in order to build groups of CAD or **CAD groups**. This should be done using the Group Editor (**Edit Groups** button). Physicals related to the spacecraft require all properties types defined (Materials, Electrical and Plasma). Physicals related to the computational volume and the external boundary require only the Plasma setting only.

Caution: An inconsistent attribution of groups will automatically lead to errors or a crash during the mapping phase or the simulation. The pertinence of the simulation is strongly dependent on a correct definition of groups.

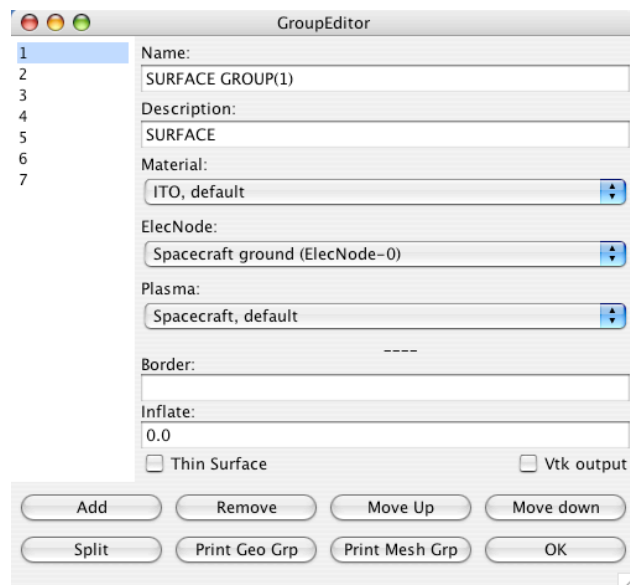


Figure 14: View of Group Editor. This Task is used to set the attribute of each group and the mapping priority. It is also used for the specific pre-processing of 2D thin elements.

The group manager allows also the possibility to re-define the priority in the properties mapping procedure. This is useful for the specific setting of elements owned by two groups. Value finally mapped will correspond to the group of higher priority. In the group manager, the priority is decreasing from the top to the bottom. The priority of a group can be changed using the **Move Up** and **Move Down** buttons. At the loading phase, groups are loaded according to the Ids of their respective Gmsh physicals.

Remark: If the group of volume corresponding to the whole computational domain has the higher priority, its values will be applied everywhere. The BC will then crashed and the system corrupted. The group of volume must always be defined in last. A wrong priority setting can lead to crash or blocking errors during the initialisation phase of the numerical kernel.

The following table gives a minimal and default setting for groups:

<i>Physical</i>	<i>Material</i>	<i>Electric</i>	<i>Plasma</i>
Spacecraft surface	ITO	ElecNode-0	Spacecraft, default
External boundary	None	None	Boundary default
Computational volume	None	None	Plasma Model in Volume, default

Groups can be visualised using the **Group Viewer** and **Cassandra** by clicking on the **Groups -> Show Groups** menu. A Cassandra viewer will be launched. Each group can be individually displayed by selection in the *View* control panel of Cassandra.

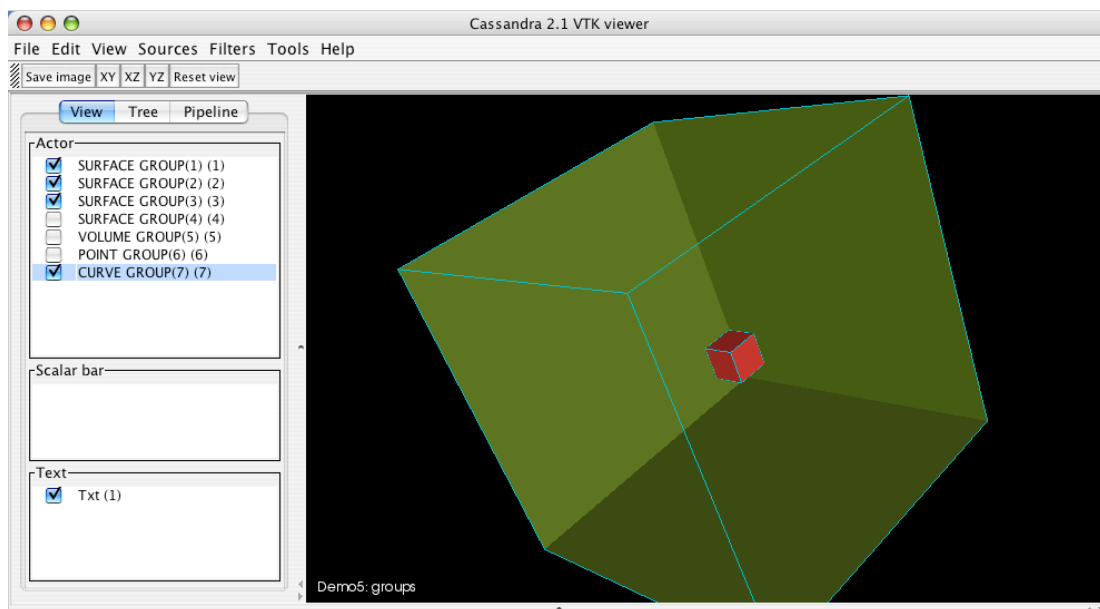


Figure 15: Visualisation of mesh groups with the Cassandra viewer.

2.5.4 Meshing

The quality of the simulation results and the stability of the solver are strongly dependent on the mesh quality. This last one may be dependent on the topology of the inner objects and the physical constraints of the modelled system.

Currently the meshing is performed using Gmsh as external tool. The resulting mesh is then loaded into the internal **Common Data Bus**. The meshing is done by clicking on the **3D Mesh** button symbolised by the following icon . You can visualise the resulting mesh through Cassandra, selecting the **Mesh->View Mesh** menu in the main menu bar.

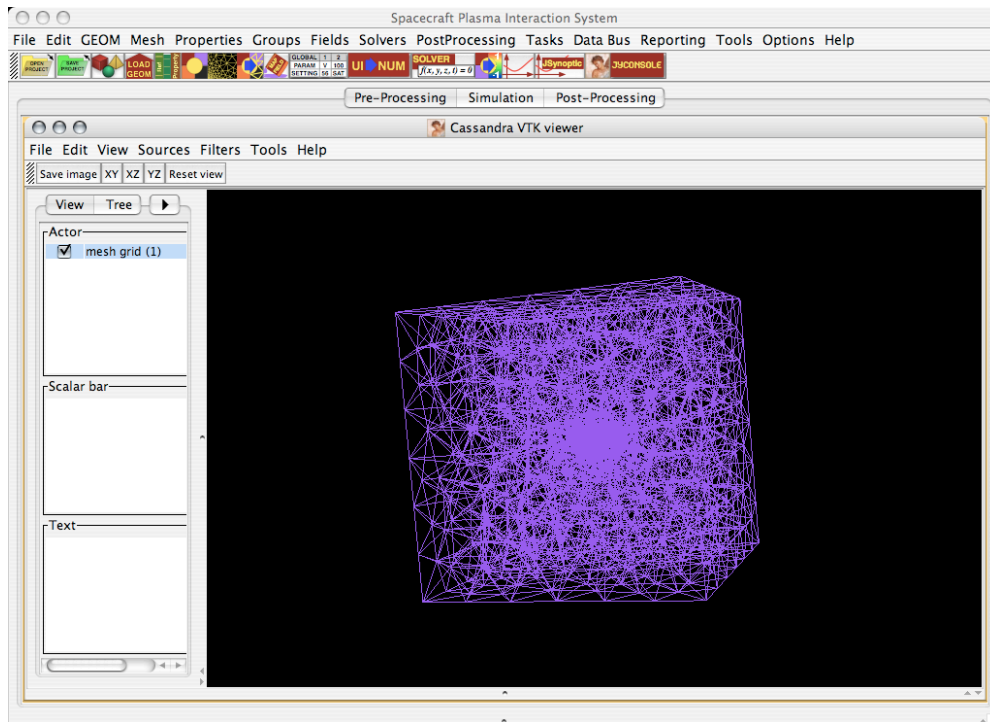


Figure 16: Visualisation of the mesh structure.

2.5.5.Groups conversion and fields mapping

For the correct deployment of local parameters, the **GEOM groups** must be converted into **Mesh Groups** before. This operation will mark individually each **Mesh Element** according to their respective GEOM groups. Marked mesh elements become **Skeletons** inside the **Common Data Bus**. The group conversion is done pushing the  icon in the tools bar or through the **Groups->Convert Groups** menu.

This will allow the Mapping of properties related to each group on the whole mesh. The fields mapping itself is done when you push on the  icon or through the **Fields->Map** menu. This phase will convert the logical attribution of various properties to each **Mesh Elements** into a set of continuous fields properly defined on the whole mesh. These fields are stocked into the **Common Data Bus** as **DataFields**. They can be visualised using the **DataField Manager** and the 3D VTK based viewers (Cassandra, Paraview...), as illustrated below.

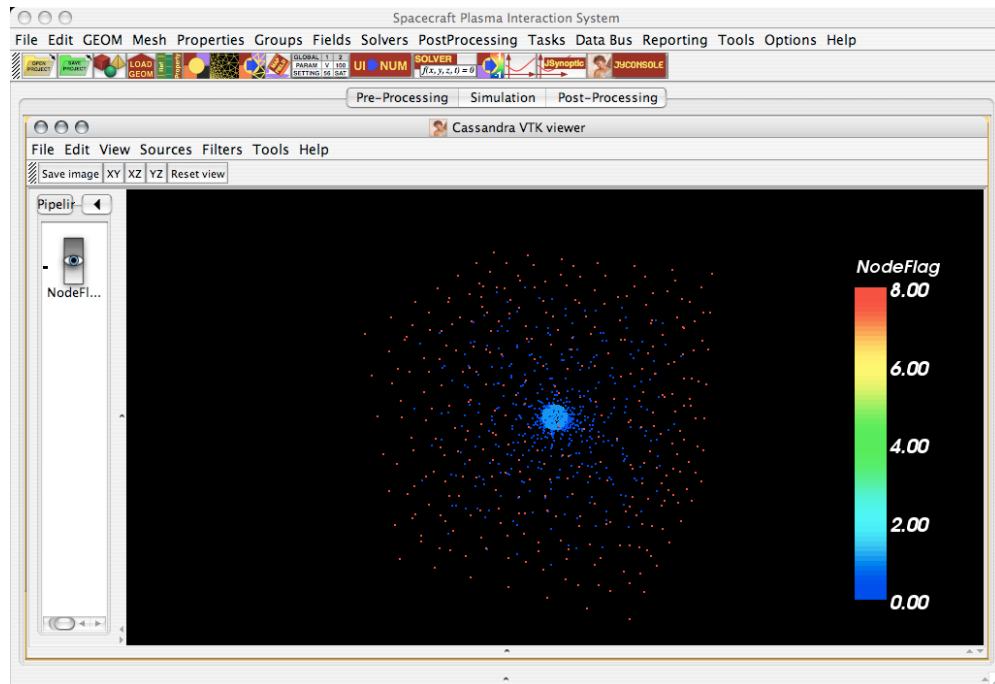


Figure 17: Example of pre-processing DataField after deployment (i.e SPIS-NUM NodeFlag). The value is set according the GEOM groups and the properties initially defined. Here NodeFlag equals 1 on the spacecraft, 0 inside the computational domain and 8 for the external boundary. The mapping is done according to the priorities of groups.

Meshing, group conversion and mapping are linked by the dependence tree of the **Tasks Manager**. Clicking on the  button will automatically call the three *Tasks* in the right order.

2.5.6 Global parameters settings

Global Parameters correspond to parameters like global physical values of the space environment (e.g. Temperature) or numerical (e.g. simulation duration). They can set using the **Global Parameters Editor** (**Global Parameters Editor** button). Many categories of parameters are defined and available through several indexes and tables. Please see the SPIS-NUM Documentation “How to control Num from UI” [13].

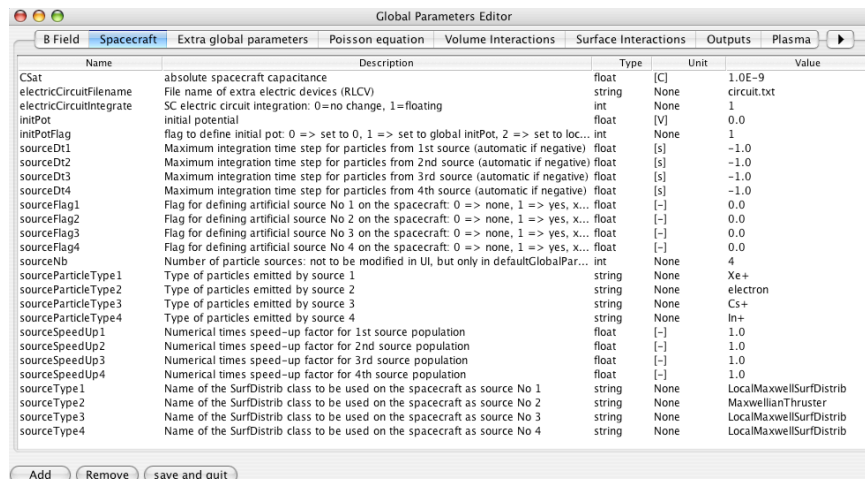


Figure 18: View of the Global Parameter editor.

2.5.7 Additional parameters of the numerical core

Some numerical kernels request specific and additional parameters, not editable with the global parameter editor, such as the definition of the internal electric circuit. These parameters are defined in text files. They can be edited with the embedded file editor by clicking on the **Solver -> SpisNum -> Edit Additional Param.**

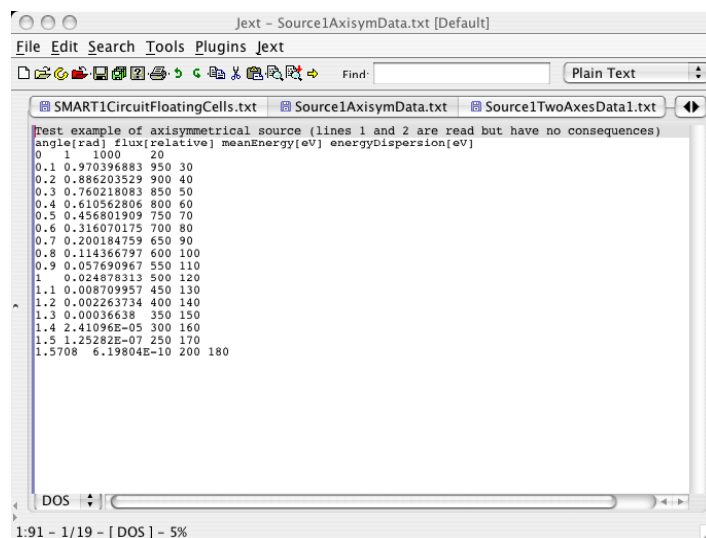


Figure 19: Edition of additional parameters with the embedded editor (Jext)

2.5.8 Final conversion UI to Num

The last step of the pre-processing procedure is the conversion of fields from the SPIS-UI data structure to the SPIS-NUM one. This Task will also build the corresponding numerical model (see the JyTop4.py script given in Annexe 3).

If a project is properly loaded at the beginning, the **UI to NUM** button will automatically call the each upstream tasks required to build the model. Error messages at this stage mean that the system definition is incorrect and must be corrected.

2.6. Numerical model generation and control of the simulation kernel

2.6.1 Multi-model approach and simulation process

The simulation runs themselves are performed by the selected numerical kernel. SPIS-UI can integrated and control several numerical kernels, with as default SPIS-NUM. All models are available through the **Solvers** menu. The simulation phase depends on the possibilities offered by the selected kernel. The description below corresponds to the use of SPIS-NUM. With SPIS-NUM, the simulation phase is typically decomposed into four phases:

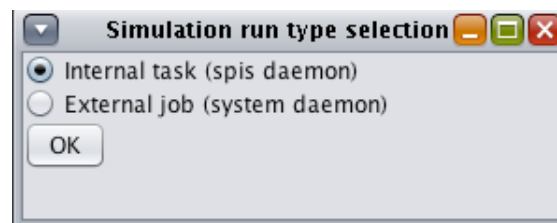
- **Init solver**, which will instantiates the solver and converts needed data.
- **Build Simulation**, which will build the whole modelling system, including the deployment of the mesh and need local fields (i.e boundary and initial conditions). At the end of this phase, the whole model is built and ready for a new simulation run. According an Object Oriented Approach, this model should be seen as a complete virtual world modelled by SPIS-NUM.
- **Run Simulation** will perform the simulation run with the pre-built model. This step can be run several times, in order to extend the total simulation time.
- **Extract Data** allows the possibility to recover computed data, i.e 3D fields and time dependent data. This action is typically done at the end of the simulation run. However, it can also be done in parallel, to extract regularly updated data during a long simulation run without waiting its issue. Then, the corresponding DataFields can be processed with the DataFieldManager (see next section).

The **Run Solver** button, with the  icon, will perform the whole simulation process and automatically these four steps.

Please see the SPIS-NUM Documentation “How to control Num from UI” [13].

2.6.2 Simulation daemon and local/remote simulation run

With SPIS-NUM, SPIS-UI (3.7 and higher) offers two different ways to launch the simulation process. The selection is done through the radio button in the dialog box as follow.



Selection dialog of the running mode for SPIS-NUM.

The running modes are:

- **As local Internal Task:** In this case, the simulation kernel is launched internally to the framework. The corresponding job is launched in an independent thread (as daemon for the SPIS-UI), in order to do not lock the framework, but in the same JVM. In this approach, the

framework keeps a pointer toward the simulation thread, allowing the possibility to dynamically interact with it. In this case, you can regularly extract updated data clicking on the **Solvers->SPIS-NUM->Extract** Data menu. In this approach, the framework should not be closed before the end of the simulation.

This approach is useful for small and medium size modelled systems (e.g typically less than $2 \cdot 10^5$ mesh elements) and during a prototyping phase. For large systems, the cost of the numerical kernel and deployed fields may become prohibitive and slow-down the framework or overtake the memory limits of your system.

Remark: Thanks to the multi-thread based design of SPIS-UI, on multi-processors/multi-core computers, the simulation thread is generally run on another node than one used for the framework.

- **In batch mode or as External Job (OS based daemon):** In this approach, SPIS-UI generates a simulation launching script, called `simulationDaemon_XXX_Run_YYY.py`, corresponding to the UI-NUM and Run Solver tasks. These scripts are stored in the root directory of the current project. They can be run independently by SPIS-UI in batch mode as follow:

```
./spis_tasks_gui.sh $PROJECT_DIR_PATH/simulationDaemon_Test_Run_8.py
```

As illustrated below, SPIS-UI can also directly run this script, just by checking the relevant check-box. In this case, you can quite the framework without stopping the simulation run.

The SPIS simulation daemon will extract the data as usual and save them in its related project. Data and results can then processed “*a posteriori*” just opening the project and loading the mesh structure and all data fields.

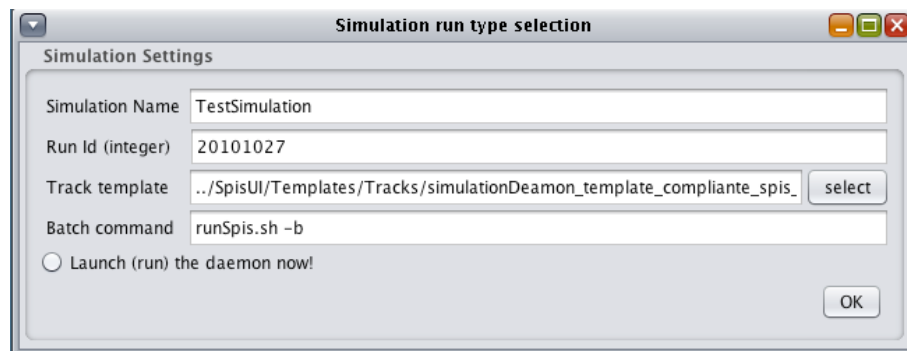


Figure 20: Settings of the simulation daemon. A simulation name and a run number (Id) must be defined. It is possible to launch directly the daemon or not.

Several track templates are available and can be selected using the select button. Please the relevant section for further information regarding the SPIS-tracks.

The daemon or “track based” approach allows also the possibility to run large simulations on a remote computer. We recommend the following approach:

- Build and model the system locally in graphical mode with SPIS-UI;
- Test the modelled system in interactive mode (i.e Internal Task) with a small system and short simulation duration;

- C. Set the real mesh size and all parameters, like simulation duration, for the realistic run and generate the simulation daemon without launching it;
- D. Tar or zip your project directory and copy it to your remote computer;
- E. Untar or unzip you project on your remote computer;
- F. Run the daemon in batch mode on your remote computer, as described above.

Remark: The mesh and the pre-processing DataField and MeshField should imperatively saved before to run the deamon.

Remark: The track file (i.e simulationDaemon_XX_Run_YY.py) should be given with its complete absolute path.

Remark: Naturally, the SPIS system must be installed on the remote computer too. Data and results can be recovered through the project as usual.

Remark: The batch mode is possible on UNIX based systems only.

2.6.3 Dynamic edition and reloading of numerical kernels

Thanks to its modular structure and introspection capability, SPIS-UI offers the specific feature to allows a dynamical edition and reloading of your numerical kernel, without restarting the whole framework.

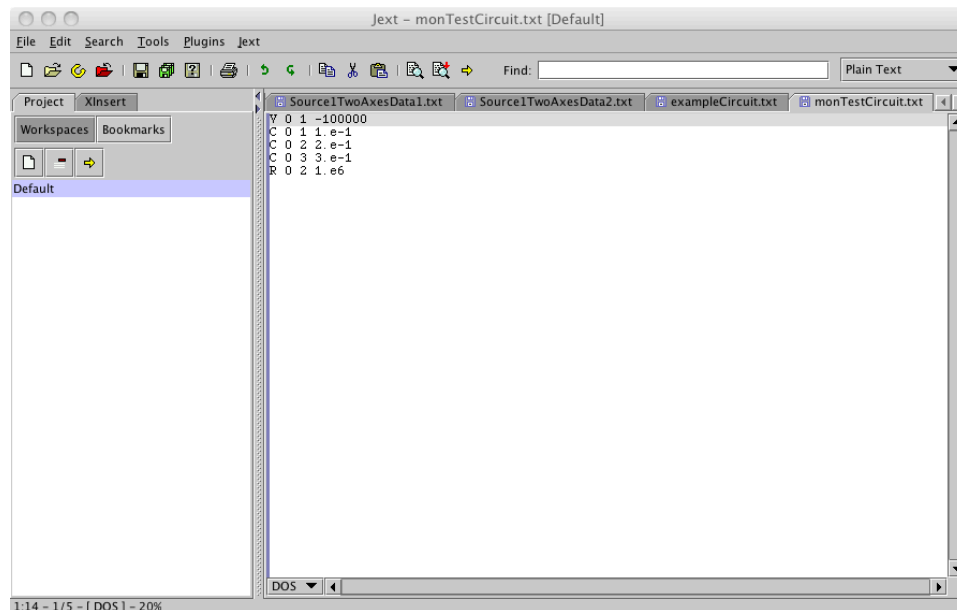
At any time during the modelling process, you can edit, modify and recompile your Java simulation kernel and reload it dynamically with the **Solvers->SPIS-NUM->Reload Solver** menu.

2.6.4 Others SPIS-NUM parameters and settings

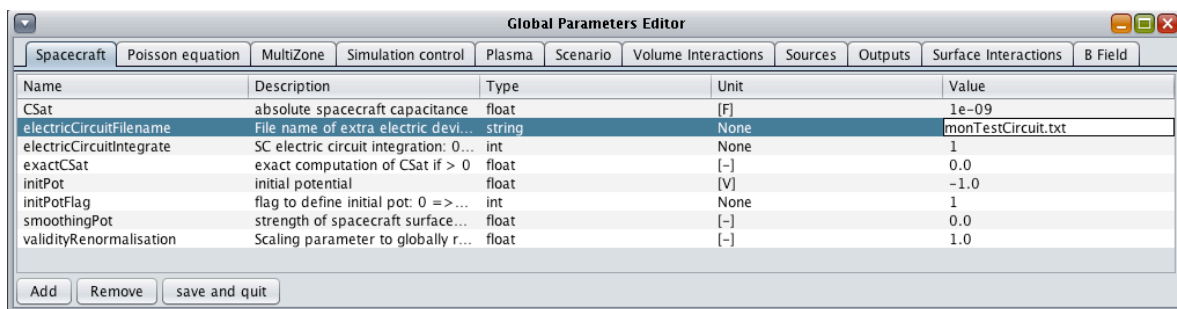
For some advanced systems (e.g. spacecraft defined with an internal circuit), additional parameters specific to SPIS-NUM may have to be edited. These are done by editing specific ASCII files.

To do this, first, please click on the **Solvers->SPIS-NUM->Edit Additional Param** menu. This will open the embedded editor (Jext) [[10](#)]. Modify all relevant files, as explained in the SPIS-NUM documentation, and saved these files must individually.

The relevant file (e.g circuit.txt) can be then defined in the global parameter. SPIS-UI will automatically managed the path and the saving process in the spis-project. The figures below show the edition these files and the corresponding setting in the global parameters.



View of the internal electric circuit definition. See SPIS-NUM documentation for further informations.



Example of the corresponding setting in the global parameter GUI. Only the name of the file should be defined. The access path is automatically managed by SPIS-UI. Please see the SPIS-NUM documentation for other files.

2.7. Post-processing and data analysis

2.7.1 3D Data analysis

All 3D **DataFields** can be converted into VTK unstructured data sets (see VTK page). This approach allows a very large set of post-processing pipelines and tools.

2.7.1.1 DataFields Manager and grid conversion

The **DataFields** conversion is done using the **DataFields Manager** (**Fields** -> **DataFields Manager** menu). You can select a **DataField** by the combo box. Several data and information are directly displayed.

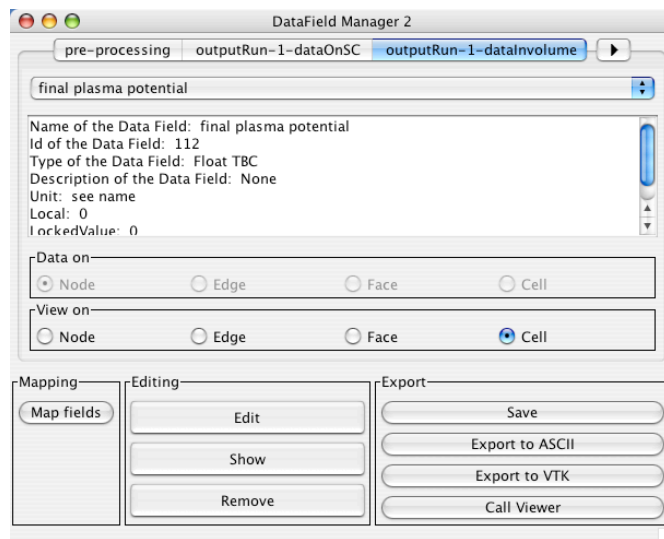


Figure 21: View of DataField Manager used to display and convert DataFields for the post-processing.

The DataField Manager allows to visualise, edit and convert the DataFields. DataFields are organised in various categories, available through a set of indexes. The **Show** action will print the values and the Id of the mesh elements associated to the DataFields.

Because the localisation on the computational grid and the type 3D representation may be different, a cell type conversion is needed. For instance, the potential computed on nodes (localisation 0) but must be displayed as continuous field on cells (localisation 3) with a linear interpolation between nodes. The DataField Manager performs these conversions. The table below summaries the possible conversions.

<i>DimOut</i> <i>DimIn</i>	0	1	2	3
0	OK			
1	OK	OK		
2	OK		OK	
3	OK			OK

The conversion into VTK unstructured data set is done by selection of the corresponding radio box and clicking on the **Export to VTK** button or is done directly

through the **Export to Cassandra** button. This will build the VTK data set and load it into the **Common Data Bus**. Output files are temporary saved into the exchange temporary directory. VTK outputs are saved into the project, when this last one is saved. The **Export to Cassandra** button will directly open a Cassandra viewer and load the data set into it. It might be needed to click on the focus icon  in Cassandra to fit the view to the loaded objects.

Remark: The export of particles trajectories into VTK format is direct by clicking on the **Export to VTK** button and does not need any cell or data structure conversion. In this case, it is not necessary to select a visualisation cell type.

2.7.1 2 3D data analysis and visualisation tools

SPIS-UI provides several 3D viewers. The 3D viewers are available through the menu **Post-processing -> 3D View**. The basic SpisViewer is now deprecated and not used anymore. Please use Cassandra or Paraview instate of the old SpisViewer.

Fig. 20 shows an example of post-processing data (Final plasma potential) displayed with Cassandra. An example of post-processing is displayed with two cutting planes and an iso-level treatment.

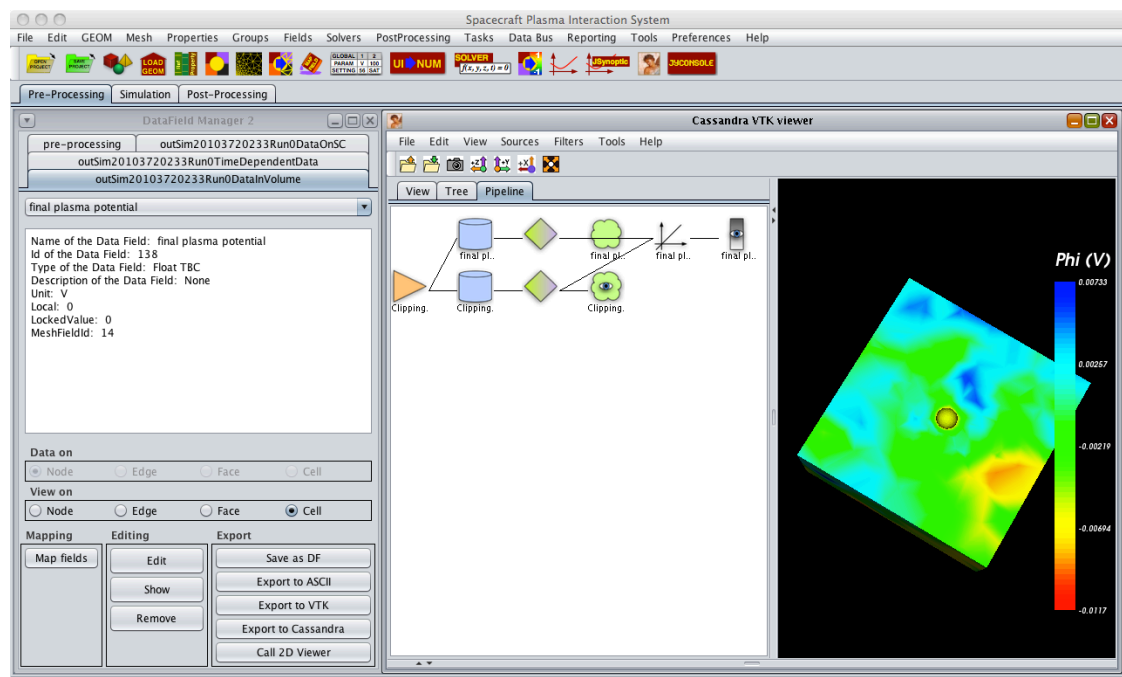


Figure 22: Example of data displayed with Cassandra in post-processing phase.

Please see the Cassandra tutorial for further information regarding 3D data analysis and post-processing [\[9\]](#).

2.7.2 2D data analysis tools and plotters

SPIS-UI provides two different viewers for $Y=f(x)$ type data, like as time dependent data.

The internal 2D viewer ( icon) is useful for standard and quick visualisation. Clicking with the right button of the mouse will open a contextual menu of setting of plot aspect and printing functions. It is also possible to zoom either by selection (left button of the mouse) or through the configuration panels. Plots can be exported as bitmap images and, on UNIX based systems, directly printed if a printer manager is set. By default, both axis are in scientific notation. Fig.21 shows a few examples of curves displayed with the 2D plot module.

For more advanced processing, the **JSynoptic** module can be used. JSynoptic can be called just by clicking on the **JSynoptic** button in the tools bar. For more details, please see the JSynoptic documentation [11].

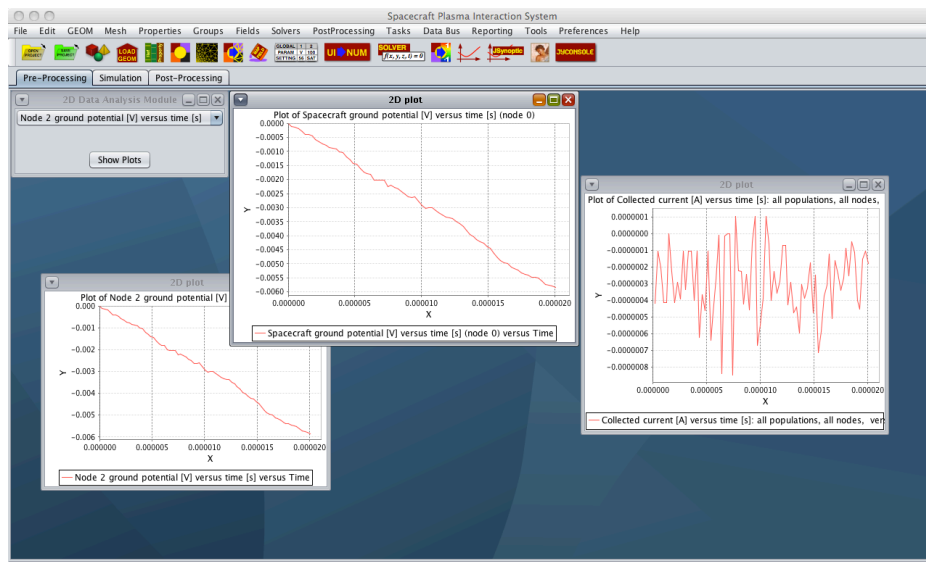


Figure 23: Visualisation of $y = f(x)$ type curves with the simple 2D plot module.

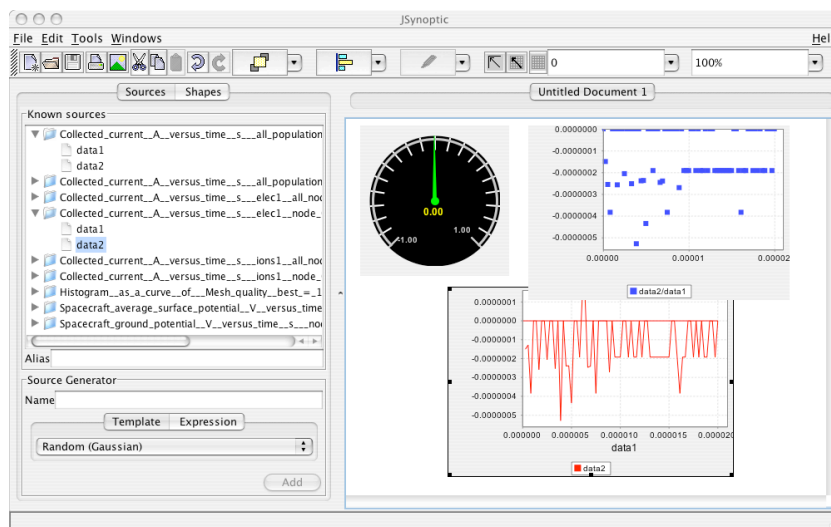


Figure 24: Visualisation of $y=f(x)$ type curves with the JSynoptic module.

2.8. Other tools

2.8.1 Tasks and Data Bus management

SPIS-UI provides a *Common Data Bus* to share all data between the various tasks. This Common Data Bus can be browsed and edited through the JyConsole as illustrated in Fig.25. This allows to directly edit, modify or remove each data present in the Common Data Bus. More information about JyConsole is available on the JyConsole Web site [7].

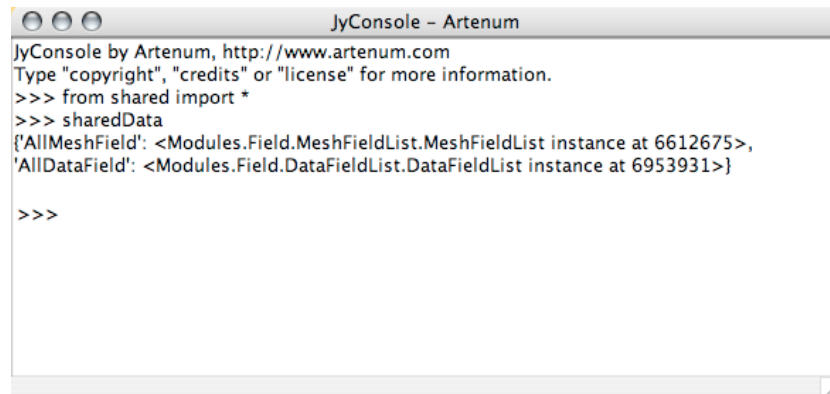


Figure 25: use of JyConsole to access to the shared data in the Common Data Bus.

The **Data Bus Manager** allows a simple access to the data, especially if you wish to remove some of items, in order to partially reset the framework. The **Data Bus Manager** is available from the **Data Bus -> Clean Data Bus** menu. An index is generated for each dictionary of the Data Bus. To reset a given data, please select it and click on the **Clean** button.

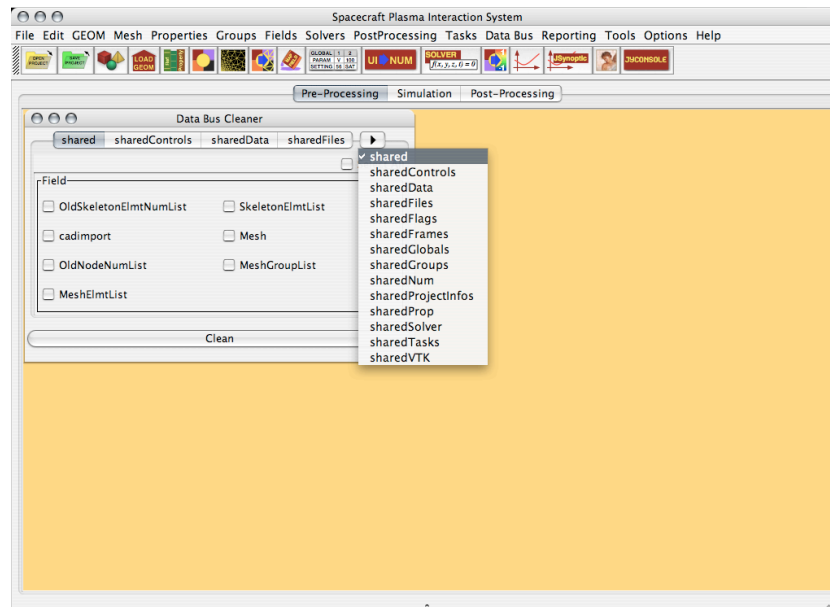


Figure 26: View of the Data Bus Cleaner

2.8.2 Memory logger

SPIS-UI provides a *set of loggers like a memory logger to observe the memory cost evolution*. The memory monitor is available through the **Tools->Memory** Monitor menu. Please take care with this function that can become costly in memory and CPU resources.

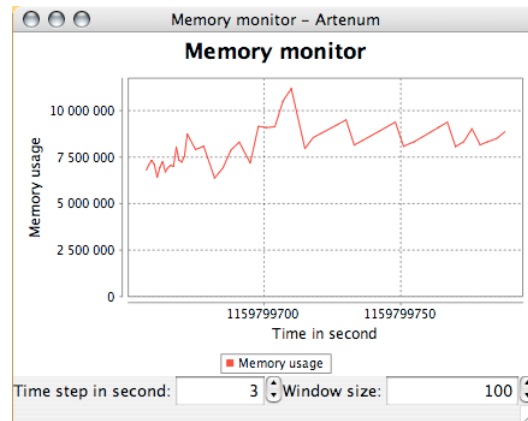


Figure 27: Example of monitors and loggers, the memory monitor. This monitor is useful to control the memory cost of large simulations.

2.8.3 Text editor and units converter

SPIS-UI includes an embedded multi-OS editor, based on the open-source project Jext. This editor can be used to edit and modify SPIS itself (scripts and source codes) and/or data and projects. The embedded editor can be called through the **Edit->File** Editor menu. Please see the TN 4.0 Developer's manual for further information [4].

SPIS-UI provides also a unit converter, able to compute the characteristic scale of your plasma. This module performs the unit conversion between the SI system and the dimensionless units system of PicUp3D project [12]. This module is still experimental and can be modified without notification.

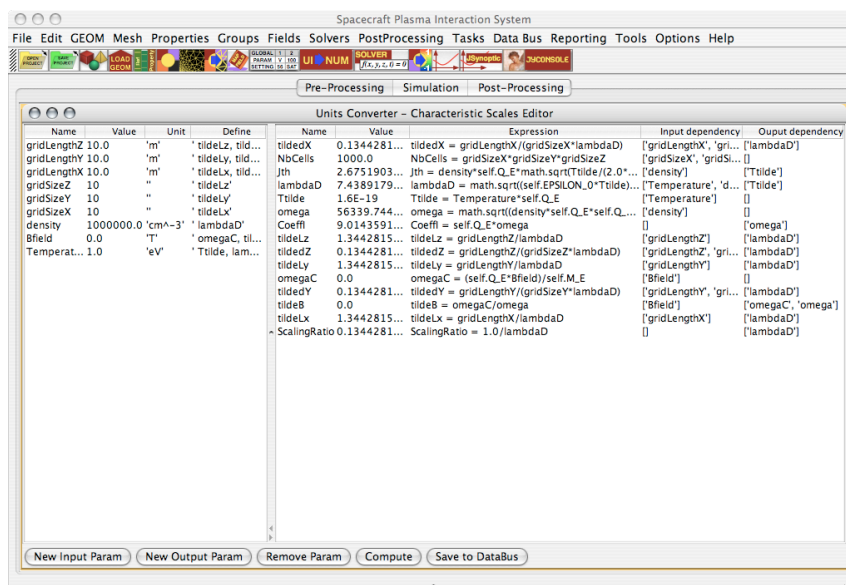


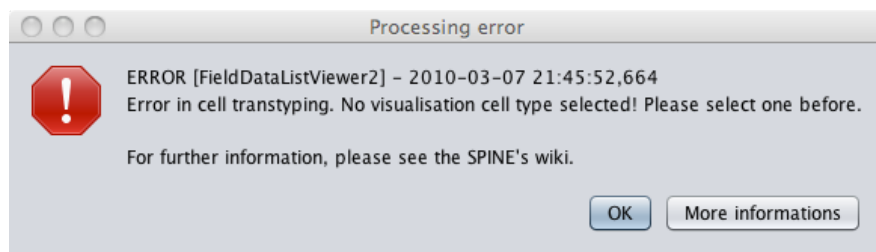
Figure 28: View of the experimental units converter (PicUp3D 3.0 project).

2.8.4 Message logger and information panel

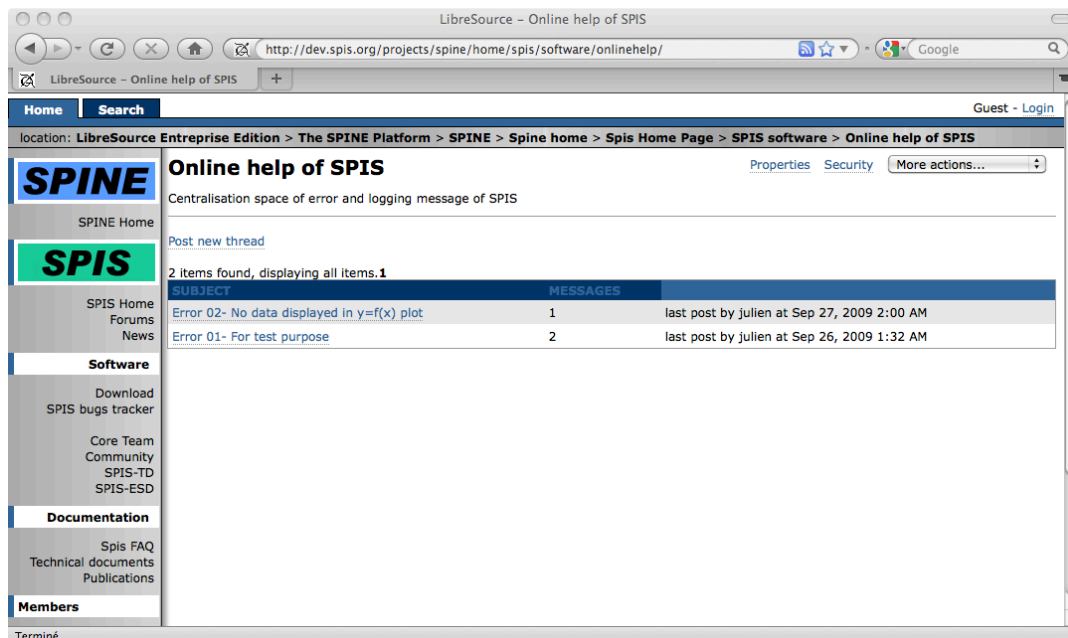
Since SPIS-UI V 4.2, an advanced system of logging has been introduced, with the possibility to select different level of verbosity. All messages are also logged in a log file saved in the project directory when the project itself is saved (otherwise this file is first written in the temporary work directory of SPIS and saved into the project when this one is saved). In case of troubleshooting or problem, we recommend to sent a copy of this file to the SPIS development team.

The various levels of message are codified using different icons and colours.

In case of locking error, an error message as-below appears. This panel tries to provide the most common reason of the problem and possible solutions. However, due to the large scale of possible configurations, all cases cannot a priori treated and a link toward a common errors/solutions wiki page on the SPINE platform is given, where solutions proposed by the community are gathered. To access to the wiki page, just click on the **More information** button. Contribution to this page are welcome.



*Example of error message panel. Click on the **More information** button to reach the Wiki page of the SPINE community, where the most common problems and solutions are gathered.*



Related wiki page where most of problem/solution couple are gathered.

The level of verbosity can be adjusted editing the \$SPIS_ROOT/SpisUI/AuxLibs/resources/log4j.xml file. Several loggers and appenders are defined, depending the logged modules and the level of verbosity can be set individually. However, it is possible to set the global level of verbosity setting the root appender at the end of the file.

```
<!--  
    Same thing could be done for the root logger, that is always build by  
    default by the logging framework. You may set here, with the priority  
    value the global level of debugging.  
-->  
<root>  
    <priority value="debug"/>  
    <appender-ref ref="console"/>  
    <appender-ref ref="file"/>  
</root>
```

Setting of the logging system.

2.8.5 Mesh Inspector and mesh quality control

In order to facilitate the meshing operation and the evaluation of the mesh quality, the mesh inspector of the JFreeMesh library, JFreeMeshInspector, has been introduced into SPIS-UI. We recommend to refer to the JFreeMesh's Web site [15] and documentation for further information.

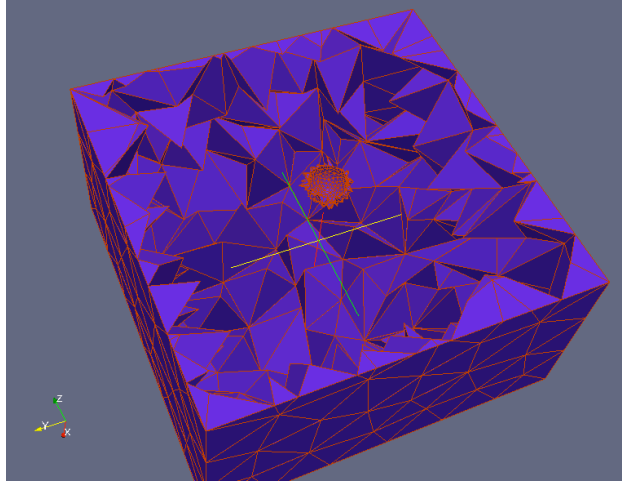
The JFreeMesh Inspector can be run as well as a standalone application than a SPIS task. In the second case, the studied mesh can be directly loaded from the Data Bus. Cassandra is integrated into the JFreeMesh Inspector to perform the 3D visualisation of result.

The present document will focus on the use of the JFreeMesh Inspector in the frame of SPIS-UI. Please see directly the JFreeMesh documentation for further information and use as standalone application.

The work logic of the JFreeMesh is the following one: For a given study (default context), we load a mesh (a study can be applied to several meshes for comparison), for the selected mesh, we create an analyser that will perform the analysis of the mesh according its predefined criterium.

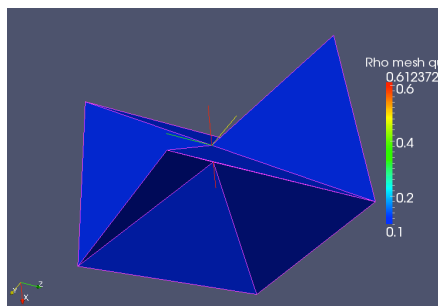
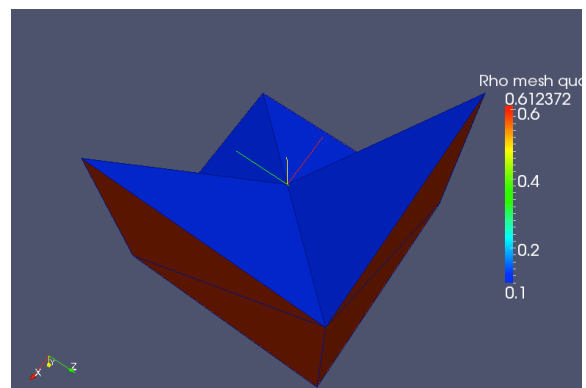
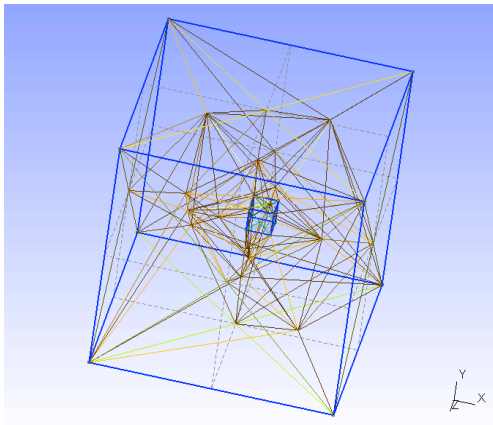
Currently, four analysers have been defined:

- **Border Cell locator**, that identify cells having on face with only one neighbour. Normally, these cells should be located on the borders of the mesh only (i.e external bound and S/C surface). An isolated or malformed cell (i.e not connected to the mesh or inside another cell) will be detected by this criterium and displayed. We recommend to use Cassandra or Paraview to perform clipping and all relevant post-processing treatments needed.



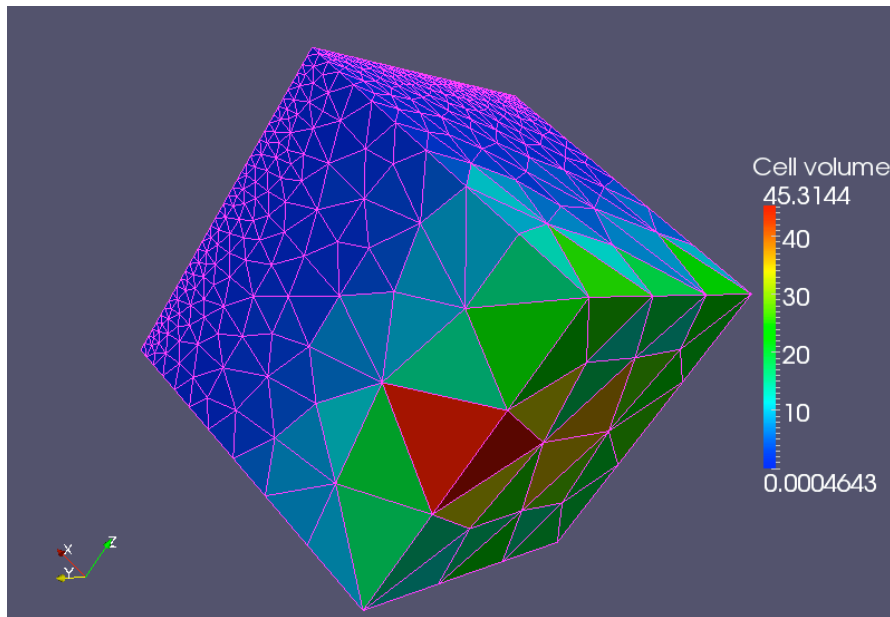
Example of analysis with the border cell locator; using the clipping filter in Cassandra On a “normal” mesh, i.e compliant with SPIS-NUM, only cells located on the borders (i.e spacecraft surface and external boundary) should appear. If a isolated cell appears inside the mesh, this last one will not be supported by SPIS-NUM.

- **Rho Quality** evaluates the ratio between the shortest and the longest edge of each cell. This will detect too “spiky” cells and will provide a good evaluation of the mesh refinement evaluation. This criterium is equivalent of the rho quality factor in Gmsh. The quality factor is normalised with respect to an isometric tetrahedron.



Example of rho quality analysis, using the VTK export and post-processing using the “threshold “ VTK filter in Cassandra or Paraview. The top left figure shows the mesh, with a voluntary poor quality. The top right picture shows the cells, where cells with a quality factor between 0.1 and 0.62. The bottom picture shows cells with a quality lower than 0.2 only. These cells corresponds the most “spiky” ones.

- **Cell Vol** returns the volume of each cell. This provides a good evaluation of the mesh refinement and homogeneity. The figure here after shows an example of cells volume analysis.



Example of cells volume analysis, viewed using the VTK export and Cassandra. Smallest cells are in blue and larger ones are in red.

- **Eta** corresponds to the same criterium as implemented in Gmsh.

Quality factors are boolean (e.g Border Cell Locator) or normalised scalar factor. They can be plotted as raw value in the console of the inspector or as histogram or as vtkDataSet structures. These last one can be loaded into Cassandra and more advanced processing, like threshold or iso-surface can be performed.

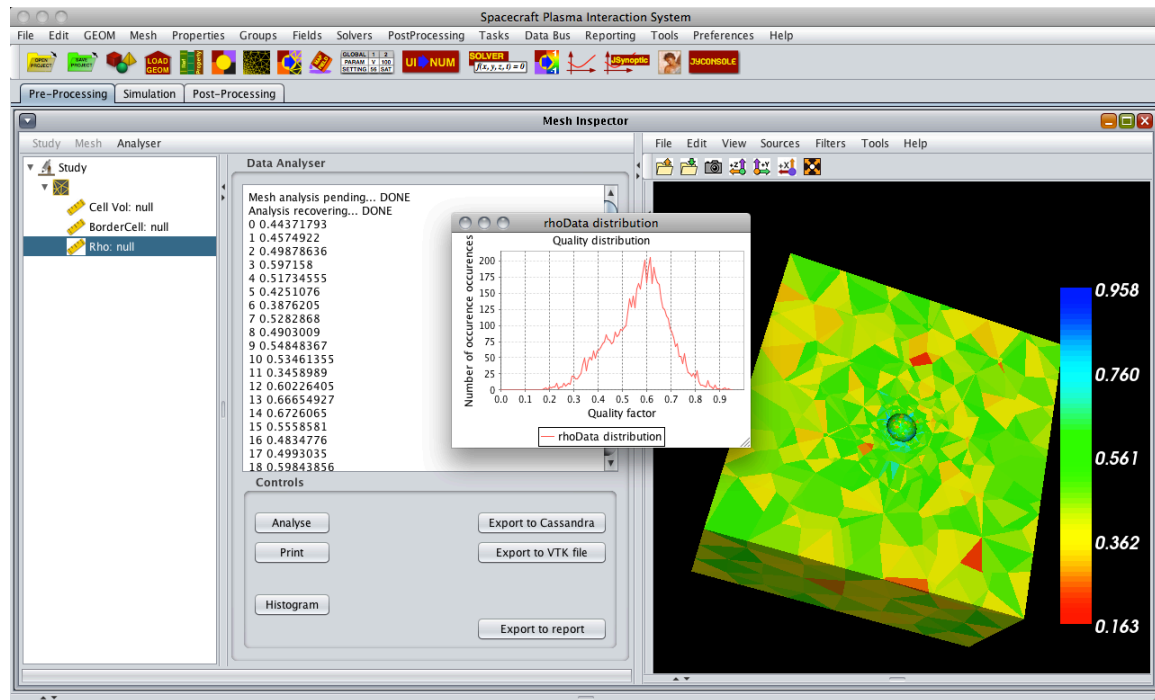
All results are saved in the current spis project in the MeshStudy sub-directory.

To load the mesh inspector, please click on the **Mesh->Mesh Inspector** menu. The mesh inspector should appear.

In the object tree on the left panel, select the study icon and click with the right button of the mouse. A contextual menu should appear, offering to load a mesh from a file or import it directly from the Data Bus.

After selection, a mesh icon should appear. Global evaluations can be done here, like computation of the mesh size, print of the meta data, evaluation of the memory cost, using the control panel.

Please then select the mesh icon in the tree and click with the right button of the mouse to add and analyser. An analyser icon should appear in the tree. The analysis can be then performed.



View of the JFreeMesh Inspector in SPIS-UI.

The JFreeMesh Inspector can also be run as a standalone application, running the `runMeshInspector.sh`. In this case, Gmsh mesh files can be read directly from the disk. The JFreeMesh Inspector accepts several command-line options. The most used is the flag calling the integration of Cassandra for the visualisation of the results, as follows:

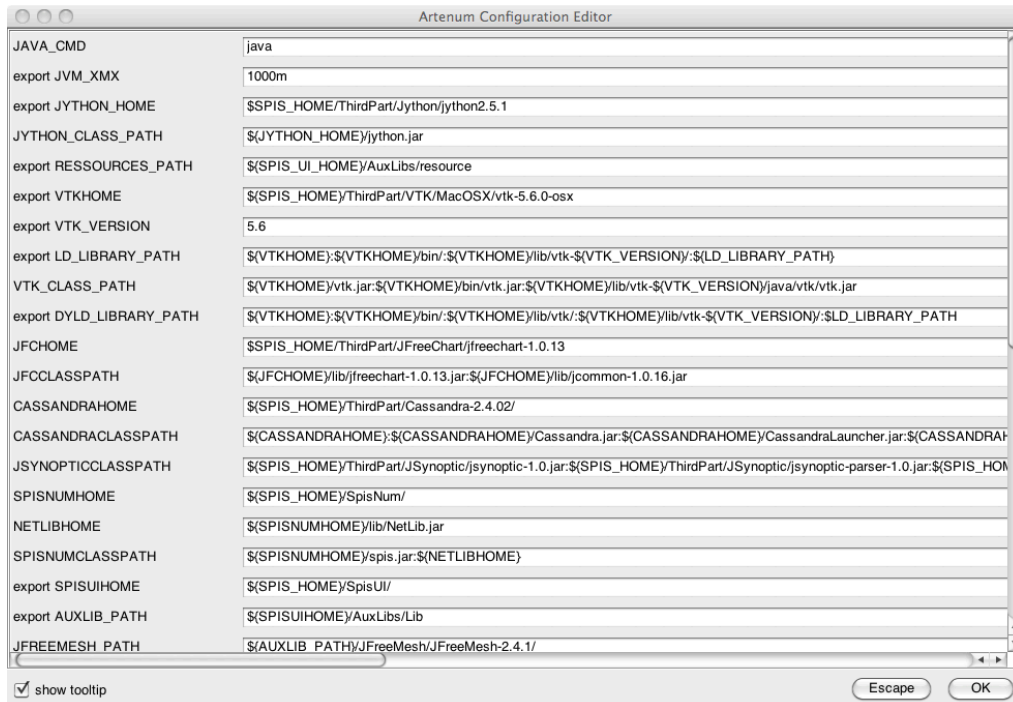
```
./runMeshInspector.sh --cassandra
```

2.8.6 Configuration and installation settings

By default, all configuration variables are set to work with the provided third-party components, including VTK and JVMs, and does not require any modification. However, if you wish to set the framework for a specific platform, e.g. optimised VTK installation or additional modules, you may have to edit:

- `$SPIS_ROOT/Bin/config.py` file: This file sets the path toward external tools such as the Gmsh modeller. It is processed at the run time (initialisation of the framework). Please notice, that some paths are OS-independent and some other may be adjusted according to your OS.
- Under UNIX-based systems, launching scripts (e.g. `runSpis.sh`) read a common properties/shell script, called `spis-ui.properties.V2`, to define all path and environment variables depending on the OS, that should be set before the execution time.

This script can be edited manually or using the graphical tool Artenum's Properties Editor, by running the `runPropertiesEditor.sh` command. The following window should appear.



View of the Artenum Properties Editor, used to set the SPIS paths and settings.

- Under MS/Windows (TM) systems, each launching scripts should be edited manually.

Please make a copy of these files before any modifications. Please see the TN 4.0 Developer's manual for further information.

2.8.7 Management and compilation tools

A set of additional tools is provided to help in the management of the projects and/or the update of the SPIS system itself.

- **duplicateProject.sh:** This command perform the copy of all elements setting a project into a new one, avoiding to copy DataFields, logs and outputs files. This is useful to develop a new simulation from an existing one. This command is available under UNIX based systems only and is used as follow:

```
duplicateProject.sh /path_to_the_project/my_old_project.spis /
path_to_new/my_new_project.spis
```

- **Ant based compilation and packaging process:** Since the version 4.3 of SPIS-UI, the whole compilation, packaging and clean-up process is based on the Apache tool Ant [14]. In normal use (i.e without modification), SPIS does not need to be re-compiled. However, as explained in after in this UM, some class files generated by Jython may have to be cleaned sometimes. This Ant based process allows also to repackage the whole application, if needed. See the Developer Manual (DM) for further information. The Ant script is run by the following command in the \$SPIS_ROOT/SpisUI directory:

```
ant
```

This will launch by default the help information as follow:

```

raminagrobis:SpisUI juju$ ant
Buildfile: build.xml

help:
[echo] SPIS-UI Ant based compilation script
[echo] - Author: Julien Forest (Artenum)
[echo] -----
[echo] Ant task available:
[echo] - build-auxlibs      : build the auxiliary librairies (see AuxLib) and install them into SPIS-UI
[echo] - build-spisnum      : build the SpisNum simulation kernel
[echo] - clean-jython       : remove all Jython $py.class files
[echo] - clean-alljars     : clean all jars
[echo] - build-all         : compile and install all jars
[echo] - release            : generate the release
[echo] - print-paths        : print the configuration paths
[echo] - spis-num-release   : generate a specific packaging of SPIS-NUM (check the version number before to proceed).
[echo] - spis-ui-release    : generate a specific packaging of SPIS-UI (check the version number before to proceed).
[echo] - doc-release        : generate a specific packaging of documentation (check the version number before to proceed).
[echo] - package-OSX-thirdPart : package the third party components for Mac OSX (Intel) platform.
[echo] - package-Linux64-thirdPart : package the third party components for Linux 64 bits (Intel)
[echo] - zip-thirdpart       : zip, date the third parties and compute the MD5.
[echo] - generate-release-header : generate the release header (i.e show all source version ticket numbers)
[echo] -----

BUILD SUCCESSFUL
Total time: 0 seconds
raminagrobis:SpisUI juju$

```

View of the Apache Ant based compilation and management process

For instance, to clean the class files generated by Jython, just run: `ant clean-jython`

2.8.8 Templates, examples projects and tests

In order to help the user several templates for key-files and example of projects are provided with the SPIS distribution.

- **\$SPIS_ROOT/Data/Templates:** This directory contains several preset project for specific application cases, like ESD modelling.
- **\$SPIS_HOME/Data/ValidationTest:** This directory contains several simple projects used as regression test during the SPIS software development. Most of them, like `test2_sphere_medium_resol.spis`, corresponds to simple cases that can be easily run and/or used as starting examples. Please see the Testing Case Technical Note for further information.
- **\$SPIS_HOME/SpisUI/Templates:** This directory gathers several sub-directories for templates files like:
 - **Geometries:** This directory contains several basic geometrical shapes supported by Gmsh.
 - **Materials:** This directory gathers all defaults NASCAP-2K and SPIS-Extended material catalogues.
 - **Tracks:** This directory gathers several tracks templates.
 - **Wizards:** this directory gathers several scenarios compliant with the Artenum's Shaman Wizard manager.

2.8.9 Online help and documentation

An extensive online help is available through the Help->Online help. This will call your Web browser and open the main documentation page. This page will give you access to whole documentation (User Manual, Developer Manual, Technical Notes, How-to, API...) and to the LibreSource platform of the SPINE community.

More generally, we strongly recommend to refer to the SPINE community forums and exchange Web platform at the following address: <http://www.spis.org>. Most of the current difficulties have been addressed by members of the community and generally several solutions and helping information are provided.

It is also possible to contact the development team at contact@spis.org and/or the various industrial and academic actors providing services of supports and teaching for SPIS.

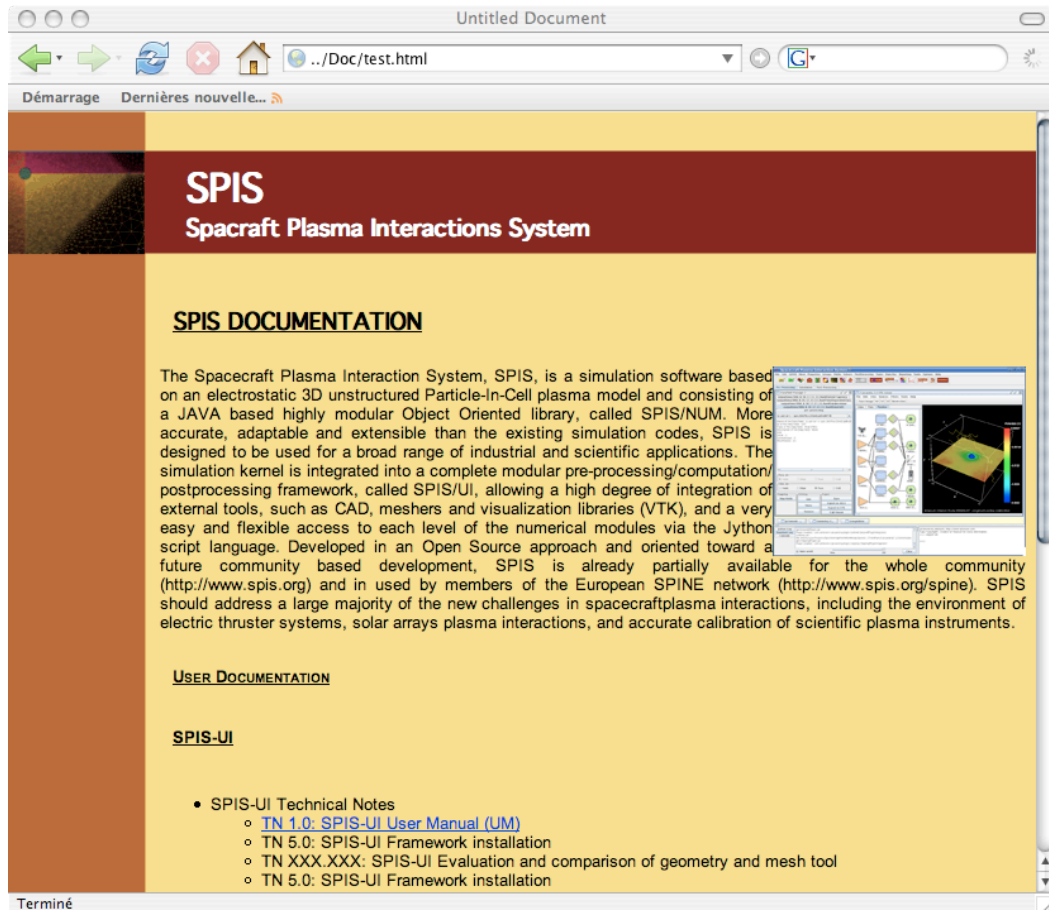


Figure 29: View of the main page of the documentation.

3 Step-by-Step how-to-drive a modelling process?

A modelling process in SPIS can be typically decomposed as follow, with no pre-existing project:

1. Create a CAD model with Gmsh, as independent tool or as integrated module. During this phase, you must defined at least three physical to define the both boundaries (inner and external) and the computational domain.
2. Load the CAD structure into the framework.
3. Set the properties, initial conditions and boundary conditions with the groups editor.
4. Mesh the computational domain using the meshing module.
5. Convert the groups and deploy the fields on the mesh.
6. Set the global parameters.
7. Convert the structure and inputs to the SPIS-NUM format
8. Perform the simulation
9. Extract and analyse the outputs results.

The Task Manager should help the user to follow this process by automatically calling for each task called by the user, all previous tasks needed for the data consistency.

This page is left intentionally blank.

4 Advanced use

4 1.Data structure, DataFields and MeshFields

Fields are stored into object oriented generic structure called DataFields. For each instance, i.e field, this class will define a name, a localisation on the cell (see below), the type of field (i.e scalar, vectors), a unit, an optional comment, an array of values and the reference to a MeshFields. The MeshField is the sub-set of the computational mesh on which one the DataField is deployed. In order to optimise the memory cost, several DataFields can refer to a same MeshField.

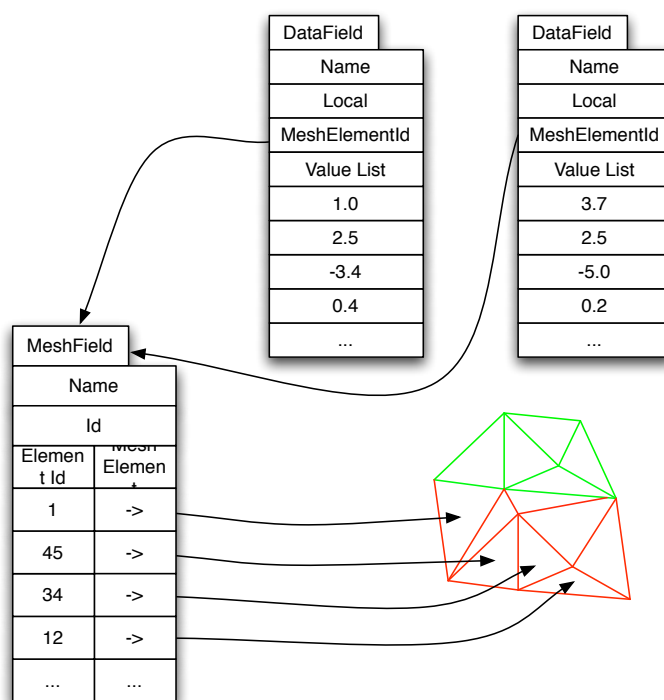


Figure 30: Principle of DataFields and MeshField.

Localisation flag	Localisation
0	Node
1	Edge
2	Face
3	Cell
4	Curvi (e.g time dependant)
5	Trajectories

DataFields and MeshFields are stored in the **Common Data Bus** and are available all the time along the modelling process. The **DataField Manager** offers a simple way to access to them and to plots their values.

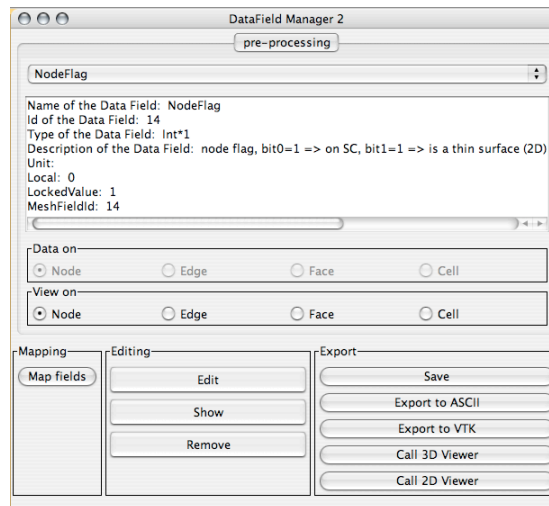


Figure 31: View of the DataFields Manager.

For more advanced use we recommend to you the Python console and directly access to the data. All DataFields are stored in the sharedData dictionary of the Bin.Task.shared module. The screen shot below show to access to the DataFields using JyConsole.

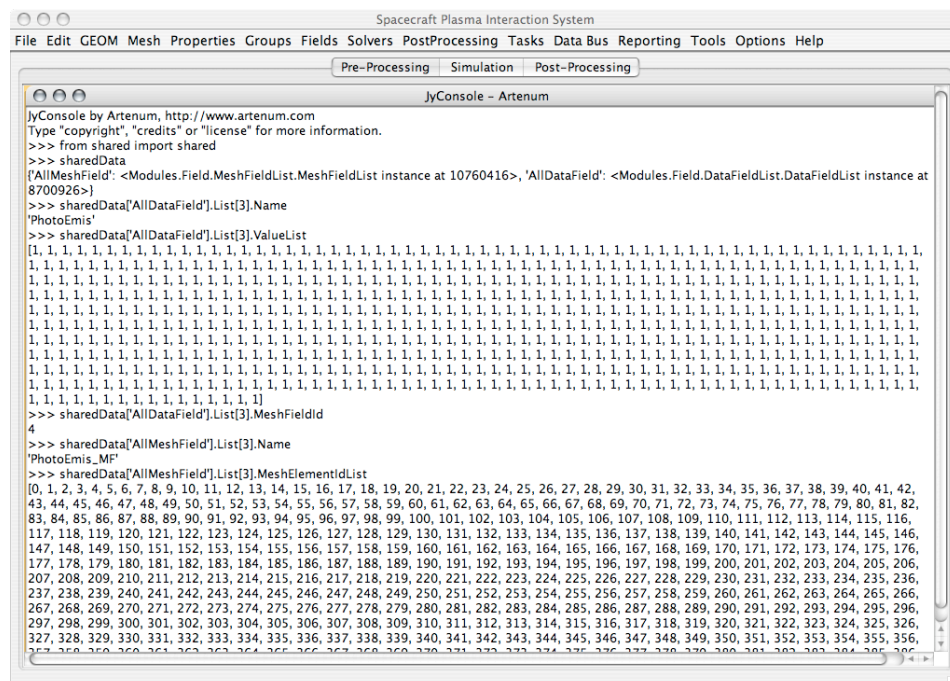


Figure 32: Access to the DataFields and MeshField using JyConsole.

To handle and access to DataFields and MeshFiels from the JyConsole, the classic approach is the following one.

- 1- Load the sharedData dictionary from the common data bus as follow:

```
>>> from Bin.Tasks.shared import sharedData
```
- 2- Recover the DataField list as follow:

```
>>> sharedData['AllDataField']
```

Use the completion function of JyConsole to select the method or the member you need. The same approach may be used for the MeshFields. For both, the access to a given DataField and MeshField can be done by its Id or its name, respectively as follow:

```
>>> sharedData['AllDataField'].GetElementById(1)
```

or

```
>>> sharedData['AllDataField'].Dic['MatModelId']
```

Remark: Because some operations may induce re-ordering in the elements, we recommend to do not use a direct access by indices in the list of DataFields or MeshFields.

3- To make the link between the DataField and the MeshField, you need to recover the Id of the MeshField supporting the DataField. This may be done as follow:

```
>>> meshFieldId = sharedData['AllDataField'].Dic['MatModelId'].MeshFieldId
```

Then, the MeshField itself can be recovered as follow:

```
>>> sharedData['AllMeshField'].GetMeshFieldById( meshFieldId )
```

Remark: A jython script, called exportDataField.py, illustrating how to access and export into an ASCII file the mesh structure and data is available in the Script directory.

4 2. Handling objects and data from the console

All objects shared in the SPIS-UI framework are seen as Jython or Java objects independently of their initial languages. They are built in a common naming space and can be manipulated directly as simple instances of objects through the Jython console (Jyconsole console) included in SPIS-UI.

4 2.1 Jython completion

With respect to the standard Jython console, **JConsole** offers an object-oriented completion. This functionality allows an easy handling Jython and Java modules and objects present in the framework.

The completion is available by pressing together the CTR+SPACE keys. An internal window must appear. All objects already present in the framework are displayed. The illustration below displays this situation. You may choose that object you want just by selection. The corresponding object will appear in the Jython console and can be directly manipulated.

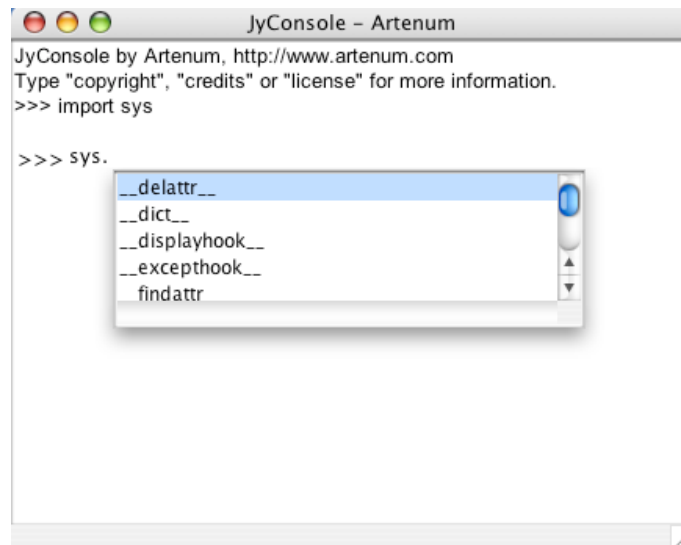


Figure 33: Illustration of the object oriented completion capabilities of JyConsole.

The completion can be performed on both Jython and Java language. This includes also native languages wrapped in Java. Be careful, the result depends if the word correspond to a module or a class (in this case only the function members are returned or if the word corresponds an instantiating of this class. In this last case, on the field members are returned.

4 2.2 Task building and execution

It is fully possible to launch all type of pre-defined tasks through the SPIS console. You only needed to choose a task with the Jython completion for example, built an instance of it as any type of Jython object and execute it with the `run_task()` method. The following example shows how to launch a Cassandra viewer through the Console:

```
>>> cas = TaskCallCassandra(Task)
>>> cas.run_task()
```

We must observe that the `TaskCallCassandra` constructor needs the `Task` object as parameter, because it is derived from this last one. The same approach can be use for Java based objects. Because the resulting instance is an independent thread, the completion gives all methods of the Python thread. The figure bellow shows the corresponding result.

The method described above instantiates and run directly tasks and modules without taking into account the dependence tree. However, tasks can be performed according the dependence tree using the *TaskManager* and passing in argument the keyword corresponding to the call task, as follow:

```
>>> from Bin.Taks.shared import sharedTasks
>>> sharedTasks['manager'].run_tasks('CADImporter')
```

For more detail, please see the SPIS-UI TN 4.0 Developer Guide [\[4\]](#).

4 2.3 Manual configuration, execution and manipulation of the SPIS-NUM kernel

The same approach can be used to set, launch and manipulate directly the solver after the pre-processing phase. This technique might be useful to build and try models and new types of simulation. The typical approach is the following one:

1. Load a project or a CAD object and define all elements needed to define the simulation (properties, groups...)
2. Maps all needed fields and define the global parameters
3. Convert the structures from UI to NUM by clicking on the corresponding button of the task bar or the Solver -> Convert UI to Num menu.
4. Load all needed fields into the data structures of SPIS-NUM using the Solver -> Init Solver menu.
5. Build the grids using the Solver -> Build Simulation menu.

At this level, all data needed by the simulation are built and stocked in the `sharedSolver['jytop']` object of the data bus. It is useful to create a dummy variable to manipulate these data more easily.

```
>>> theTop = sharedSolver['jytop']
```

All fields are members of this object and can be handle using the completion.

To create a new simulation kernel, you need to import the corresponding Java class as follow:

```
>>> from spis.Top.Simulation import *
```

Path to the various inputs files might be also necessary:

```
>>> import Bin.config
```

```
>>> from Bin.config import GL_EXCHANGE, GL_DEFAULT_INPUT_PATH, GL_DATA_PATH
```

The simulation object can be now built and initialised, as follow, for example:

```
>>> theTop.simu = SimulationFromUIParams( theTop.volMesh,\
                                         theTop.bdSurfMesh,\
                                         theTop.scSurfMesh,\
                                         theTop.globalParameterArray,\
                                         theTop.localParameterArray,\
                                         GL_DATA_PATH,\
                                         GL_DEFAULT_INPUT_PATH)
```

The detailed syntax depends on the constructor. This object can be manipulated link the corresponding Java object. The execution of the simulation loop is simply done as follow:

```
>>> theTop.simu.integrate()
```

After creation of the simulation object, it is possible at each step to recover at each step all data and generate the output DataFields just by clicking on the **Solver->Solver->ExtractData** menu and using the DataField manager, as usual.

4 2.4 SPIS-UI in batch mode and use of SPIS's tracks

SPIS-UI is fully scriptable using Jython based scripts called *Spis Tracks*. *Spis Track* are useful either to run SPIS in batch mode, e.g on a remote server without X11 export, or to perform specific modelling/simulation processes, like parametric studies, data-mining, post-processing, etc... As all Jython scripts, Spis Tracks can be edited manually, but we recommend to start from templates provided in the \$SPIS_ROOT/SpisUI/Templates/Tracks directory. This is the case, for instant, when SPIS-NUM is run in *External Daemon* mode. An example of track is given in annexe 2.

Important remark: Spis Tracks earlier than SPIS 4.3 will not be supported by higher version of SPIS. Since SPIS 4.3, a version number (defined by the GL_TRACK_VERSION variable) should be set at the beginning of each track. In case of disfunction of the tracks loader, please this version number.

Important remark: The track loader supports two modes of processing of tracks. the inlined one (when IN_LINED_TRACK equal to 1), where the track is processed in only one Jython exec process. This mode provide the best performances and allows indented function (like for loops). However, in case of error, the break point will not be returned. In non-inlined mode, the track is processed line by line. For now, this does not allow indented functions, but this mode will return the line number in the track script in case of error during the process.

SPIS-UI tracks can be run in several manners:

1) **In interactive mode through the GUI:** It is possible to directly run the tracks in the GUI mode, to test-them or perform specific processes, for instance. To do it, click on the **File -> Load Spis Track** menu. A file chooser should appear. The track is processed as soon as it is selected. The outputs are generally printed in the standard logging window.

2) **As script in a JyConsole:** To load a script through JyConsole, you can use the classic import command of Python or click in the area of this console with the right button of your mouse. A contextual menu must appear with the load script command.

3) **As batch script:** Most of the SPIS-UI tracks can be run in batch mode as follow:

```
spis_task_graph.sh -b path_to_myscript/myscript.py
```

Remark: In all cases, the loaded script must be in the Python path.

Remark: The track file (i.e myscript.py) should be given with its complete absolute path.

4 3. Supported platforms

4 3.1 Java and native layers : handling of the full or degraded modes

SPIS is Java based and is normally runnable on all platform (i.e hardware/OS couple) support a Java runtime following the SUN's specifications for the JSE.

The minimum recommended version of the JVM is 1.6 and higher.

Only native components (e.g Gmsh, VTK, Paraview) may require a specific packaging or re-compilation for each platform.

By default, SPIS is packaged for the most common platforms:

- Linux in 32/64 bits;

- Apple Mac OSX 32/64 bits;
- Windows 32 bits.

In these case, SPIS is package with the relevant JRE and native components.

If the native components are not supported on the local machine, SPIS can generally be run in degraded mode. In this case some functions can be not supported (e.g 3D visualisation layers, meshing capabilities). However, the simulation loop and kernel is still fully operational.

NB: Please take care, that some 3D visualisation components (e.g VTK) may depends on your graphics configuration (i.e video card and driver) and some external libraries (e.g OpenGL and/or DirectX). If specific configurations have been done previously on your computer (e.g another installation of VTK), please take care that their is not risk of conflicts through the environment settings, for instance.

4 3.2 Memory cost and limitations

3D PIC models and mesh analysis may be intensive in memory and CPU resources. A proper setting of your JVM might be critical to run your models.

The recent JVMs (1.6 and higher) have remove memory limitation on most of the platforms. Several runs have been successfully performed with more than 20Gb of RAM (i.e about 10^6 tetrahedrons mesh in full PIC mode with about 15 particles by cells) on UNIX/Linux 64 bits systems (Java 1.6 Sun).

The maximum addressable memory size can be set by editing edit the file called spis-ui.properties in the SpisUI/BinLinux directory (if you are under Linux). For Windows, you should edit the runXXX.bat script, where XXX is the requested application (e.g SPIS).

Just adjust the value of value of the memory Java stack size, i.e the variable JVM_XMX, as follow in the section :

```
#####
# Setting of the memory Java stack size          #
#####
export JVM_XMX=1000m
```

This will actually set the Xmx property of the JVM. By default, it is set to 1Gb in order that most of users can at least start SPIS on small computers.

Set it to 4Gb (i.e JVM_XMX=4000m) or more if you computer has more RAM. This should still be lower than the real total RAM.

On a normal laptop (Mac OSX with 4Gb), it is normally possible to run easily systems with up to $9.E5$ tetrahedrons.

IMPORTANT REMARK: Above comments are valuable only for UNIX based system. Under most of standard desktop Windows system, the total amount of memory dedicated to one job is limited to less than 2Gb. In this case, it is impossible to overtake this limitation due to the OS.

This why, we strongly recommend for large modelled systems to perform your simulations on UNIX based systems or Windows server OS.

Remark: The total memory cost may also depend on the used JVM and its settings. Please see the JVM's documentation for further information.

4 3.3 Difference between 32 and 64 bits systems ?

Excepted for the VTK native layer, for which one a compilation is of course needed each platform, there is normally not difference of behaviour between a run on a 32 bits and the 64 bits platform. In the worst case, this should only impact the UI layer (e.g Swing look and feel, I/O access and performances, JNI layer) or the whole memory cost, but not the simulation loop (i.e results of NUM).

At least, this is one of the specification of JVMs by SUN. And the JVMs provided with SPIS are SUN's ones.

Maybe three cases may have to be checked, however:

1) We cannot guaranty this if a different local JVM not following the whole SUN's specifications is used.

2) In SpisUI and in the projects, it is recommended sometimes to clean up the `*$py.class` files. These files correspond to the Python scripts compiled and converted into class file by Jython to be run. In some cases (e.g old RedHat distributions for instance), we observed that these classes maybe not properly updated in spite of the Python's scripts have been modified and are newer. In this case, the framework may restart on the old classes and the data corrupted.

Just remove the `*$py.class` files (script `clean.sh` in SpisUI) and re-run the framework.

5 References

- [1] Python Community Web Site, <http://www.python.org>
- [2] SPINE Community's Virtual Lab, <http://www.spis.org>, 2002
- [3] Cassandra's Web Site, <http://www.artenum.com/cassandra>
- [4] Forest J., *SPIS-UI Documentation: TN 4.0 Developer Guide*, SPIS project 2006
- [5] Gmsh User Tutorial for CAD and Meshing
- [6] Geuzaine C., Remacle J.-F., *Gmsh User Guide*, <http://geuz.org/gmsh/#Documentation>
- [7] JyConsole's Web site, <http://dev.artenum.com/jyconsole>
- [8] Jython Community Web Site, <http://www.jython.org>
- [9] Jourdain S., Cassandra's tutorial, <http://www.jython.org>
- [10] Jext's Web site, <http://www.jext.org/>
- [11] Jsynoptic's Web site, <http://nicolas.brodu.free.fr/fr/programmation/jsynoptic/index.html>
- [12] Picup3D Project's Web site, <http://dev.spis.org/projects/spine/home/picup>
- [13] Roussel J.-F., *SPIS-NUM Documentation : Controlling Num from UI*, SPIS Project, 2006
- [14] Apache Ant project's Web page, <http://ant.apache.org/>
- [15] JFreeMesh's Web site, <http://dev.artenum.com/projects/JFreeMesh>, 2003

This page is left intentionally blank.

6 Annexe 1: Launching scripts

6 1.Unix like configuration scripts

NB: Since SPIS 4.2, these configurations elements are set in the spis-ui.properties file in place of the launching scripts themselves.

```
#!/bin/bash
# Modify the line above to match your shell (must be a bourne-shell).
#####
#
#      MAIN SCRIPT OF LAUNCHING OF THE SPIS FRAMEWORK      #
#
# Please see the Readme.txt file for settings and uses      #
#
# (c) Artenum, Paris, 2003-2004                              #
#####
echo "Please, wait... "
cat ./Version

# If the following settings are not accepted by your shell,
# please set the corresponding environment variables and comment
# the following lines excepted teh las one.

#####
# not mandatory. Needed only if you want to launch Spis    #
# from an icone in the KDE desktop or on in stand-alone     #
#####
HERE=.
SPIS_HOME=$(pwd)/..
export JVM_HOME=$SPIS_HOME/ThirdPart/JVM/Linux-I386/
#export JVM_HOME=/user/shared/j2sdk1.4.2_06/

#####
# Setting of the Jython interpreter                          #
# Please set the path of your Jython interpreter            #
#####
export JYTHON_HOME=$SPIS_HOME/ThirdPart/Jython/jython-2.1
JYTHON_CMD=$JYTHON_HOME/jython
export CLASSPATH=$JYTHON_HOME/jython.jar:$CLASSPATH

#####
# Setting of the VTK related elements                        #
# Please set these paths according you own installation     #
# (i.e. VTKHOME is where is the root directory of your VTK #
# installation                                              #
#####
# if you use the embeded VTK stuff in ThirdPart
export VTKHOME=./ThirdPart/VTK/I386-Linux/VTK/
# otherwise, if you have a VTK stuff already installed...
#export VTKHOME=/Users/juju/VTK/VTKOSX/CompilationSpace/
export LD_LIBRARY_PATH=${VTKHOME}/bin/:${LD_LIBRARY_PATH}
export CLASSPATH=${VTKHOME}/bin/vtk.jar:$CLASSPATH
export DYLD_LIBRARY_PATH=${VTKHOME}/bin/:${VTKHOME}/lib/vtk/:${LD_LIBRARY_PATH}

#####
# To access to the "Wrapping/Java" equivalent files        #
# Settings by default to the local version compiled with    #
# JVM 1.3                                                    #
# export CLASSPATH=./vtk_local/vtk1.3/:$CLASSPATH           #
```

```

# otherwise comment the previous line and use you onw      #
# VTK installation as follow                                #
#####
export VTKJAVAWRAPPING=${VTKHOME}/Wrapping/Java
export CLASSPATH=${VTKJAVAWRAPPING}:${CLASSPATH}

#####
# Setting of the JFreeChart Lib                              #
# Normally do not need any modifications                    #
#####
export JFCHOME=../ThirdPart/JFreeChart/jfreechart-0.9.16/
export JFCCLASSPATH=${JFCHOME}/jfreechart-0.9.16.jar:${JFCHOME}/lib/jcommon-0.9.1.jar:${JFCHOME}/lib/servlet.jar

#####
# Setting of the Cassandra viewer                            #
# Normally do not need any modifications                    #
#####
export CASSANDRAHOME=${SPIS_HOME}/ThirdPart/Cassandra-2.2-jdk1.4-vtk4.2
export CASSANDRACLASSPATH=${CASSANDRAHOME}:${CASSANDRAHOME}/Cassandra.jar:${CASSANDRAHOME}/CassandraLauncher.jar:${CASSANDRAHOME}/ThirdPart/skinlf/skinlf.jar:${CASSANDRAHOME}/thirdpart/DedaleGraph/DedaleGraph.jar:${CASSANDRAHOME}/thirdPart/Jyconsole/JyConsole.jar:${CASSANDRAHOME}/thirdpart/CassandraPipelineManager-1.1/ArtenumPipeLineManager-1.1.jar
export CLASSPATH=${CASSANDRACLASSPATH}:${CLASSPATH}

#####
# Setting of the JSynoptic                                  #
# Normally do not need any modifications                    #
#####
export CLASSPATH=${SPIS_HOME}/ThirdPart/JSynoptic/jsynoptic-1.0.jar:${SPIS_HOME}/ThirdPart/JSynoptic/jsynoptic-parser-1.0.jar:${SPIS_HOME}/ThirdPart/JSynoptic/simtools-1.0.jar:${SPIS_HOME}/ThirdPart/JSynoptic/syn3d-1.0.jar:${CLASSPATH}

#####
# Setting of the SPISNUM module (by default, should not be #
# modified excepted in some specific cases)                #
#####
export SPISNUMHOME=${SPIS_HOME}/SpisNum/
export NETLIBHOME=${SPISNUMHOME}/lib/NetLib.jar
export CLASSPATH=${SPISNUMHOME}/spis.jar:${NETLIBHOME}:${CLASSPATH}

#####
# Setting of the jModule (Normally do not need any        #
# modifications.                                           #
#####
export JMODULEPATH=./jModule/jModuleUI.jar:./Lib/JFreeMesh/JFreeMesh.jar:./Lib/JFreeMesh/JFreeMesh-VtkBuilder.jar
export CLASSPATH=${JMODULEPATH}:${CLASSPATH}

#####
# Setting of the GUI modules                                #
#####
export GUIPATH=$HERE/GUI/SpisGui.jar:$HERE/GUI/lib/ArtTk.jar:$HERE/GUI/lib/JyConsole.jar
export CLASSPATH=${GUIPATH}:${CLASSPATH}
export CLASSPATH=./GUI/Icons:${CLASSPATH}

#####
# Setting of the postprocessing modules                    #
#####
export POSTPROPATH=$HERE/Lib/FieldManagerUI/lib/fieldManagerPanelPkg.jar
export CLASSPATH=${POSTPROPATH}:${CLASSPATH}

```

```

export SPIS2DPLOTPATH=${SPIS_HOME}/SpisUI/Lib/Spis2DPlot/lib/spis2Dplot.jar
export CLASSPATH=${JFCCCLASSPATH}:${SPIS2DPLOTPATH}:${CLASSPATH}

#####
# Setting of the Documentation Viewer (by default, should #
# not be modified excepted in some specific cases)      #
#####
export CLASSPATH=${SPIS_HOME}/ThirdPart/Multivalent/Multivalent20040415.jar:${CLASSPATH}

#####
# Setting of the jext editor                               #
#####
export JEXTPATH=${SPIS_HOME}/ThirdPart/Jext/
export CLASSPATH=${JEXTPATH}/lib/dawn.jar:${JEXTPATH}/lib/jext.jar:${CLASSPATH}

export JYTHONARCH=/tmp/jythonArch-$USER/
mkdir $JYTHONARCH &>/dev/null

#####
# This is the launching line (do not modify it).         #
#####
exec $JYTHON_CMD -i -Dpython.cachedir=$JYTHONARCH -Dpython.path=${SPISNUMHOME}:${SPIS_HOME}/
SpisUI/Modules/Adapter:$SPIS_HOME/SpisUI:$SPIS_HOME/SpisUI/Utils/spisUtil.jar:$SPIS_HOME/
SpisUI/PostProcessing/Lib/SpisViewer2.jar:${VTKJAVAWRAPPING} ${SPIS_HOME}/SpisUI/Bin/Tasks/
SpisTasksGraph.py $@

```

6 2.Windows

```

@echo off
REM #####
REM #
REM #   Artenum http://www.artenum.com   #
REM #
REM #   Windows script, path must be           #
REM #   configurate for system                 #
REM #
REM #####

REM #####
REM #   Start of the variable which be         #
REM #   configurate                           #
REM #####

set SPISPATH=..
set SPISPATHR=..\..\..\
set JYTHONPATH=%SPISPATH%\ThirdPart\Jython\jython-2.1\
set JYTHONPATHR=%SPISPATHR%\ThirdPart\Jython\jython-2.1\
REM echo %JYTHONPATH%
set JYTHON_HOME=%JYTHONPATH%
set VTKPATH=..\..\..\ThirdPart\VTK\Windows\vtk42
set JAVAPATH=..\..\..\ThirdPart\JVM\windows\jre\

REM #####
REM #   End of the variable which be           #
REM #   configurate                           #
REM #####
more %SPISPATH%\SpisUI\Version
echo Please, wait...

set CLASSPATH=%JYTHONPATHR%\Lib;%JYTHONPATHR%\jython.jar;%CLASSPATH%
set PYTHONPATH=%JYTHONPATHR%\Lib;%PYTHONPATH%

```

```

set PATH=%PATH%;%JYTHONPATH%

set PATH=%PATH%;%JAVAPATH%\bin;%JAVAPATH%\jre\bin;%VTKPATH%\bin\vtk.jar;%VTKPATH%\bin\;%VTKPATH%
set LD_LIBRARY_PATH=%VTKPATH%\bin;%VTKPATH%\lib

set CLASSPATH=%VTKPATH%\bin\vtk.jar;.;%JYTHONPATH%;%SPISPATHR%\SpisUI\vtk_local\src;%SPISPATHR%\SpisUI\GUI\SpisGUI.jar;%SPISPATHR%\SpisUI\GUI\lib\ArtTk.jar;%SPISPATHR%\SpisUI\GUI\lib\JyConsole.jar;%SPISPATHR%\SpisUI
set PYTHONPATH=%CLASSPATH%;%SPISPATHR%\SpisUI;%SPISPATHR%\SpisUI\Bin

REM #####
REM # Setting of the SPISNUM module (by default should ne be #
REM # modified) #
REM #####
set CLASSPATH=..\..\..\SpisNum\spis.jar;%CLASSPATH%
set NETLIBHOME=..\..\..\SpisNum\lib\NetLib.jar
set CLASSPATH=%NETLIBHOME%;%CLASSPATH%

REM #####
REM # Path for the multi-valent browser #
REM #####
set CLASSPATH=..\..\Multivalent\Multivalent20040415.jar;%CLASSPATH%

REM #####
REM # Setting of the Cassandra 2.1 viewer #
REM # Normally do not need any modifications #
REM #####
set CASSANDRAHOME=%SPISPATHR%\ThirdPart\Cassandra-2.2-jdk1.4-vtk4.2
set CASSANDRACLASSPATH=%CASSANDRAHOME%;%CASSANDRAHOME%\Cassandra.jar;%CASSANDRAHOME%\CassandraLauncher.jar;%CASSANDRAHOME%\ThirdPart\skinlf\skinlf.jar;%CASSANDRAHOME%\thirdPart\Dedalegraph\Dedalegraph.jar;%CASSANDRAHOME%\thirdPart\artenum\Jyconsole\JyConsole.jar;%CASSANDRAHOME%\thirdPart\CassandraPipelineManager-1.1\ArtenumPipeLineManager-1.1.jar
set CLASSPATH=%CASSANDRACLASSPATH%;%CLASSPATH%

REM #####
REM # Setting of the GUI modules #
REM #####
set GUIPATH=%SPISPATHR%\SpisUI\GUI\SpisGui.jar;%SPISPATHR%\SpisUI\GUI\lib\ArtTk.jar;%SPISPATHR%\SpisUI\GUI\lib\JyConsole.jar;%SPISPATHR%\SpisUI\GUI\Icons
set CLASSPATH=%GUIPATH%;%CLASSPATH%
set CLASSPATH=%SPISPATHR%\SpisUI\GUI\Icons;%CLASSPATH%

REM #####
REM # Setting of the JSynoptic #
REM # Normally do not need any modifications #
REM #####
set JSYNOPTICPATH=%SPISPATHR%\ThirdPart\JSynoptic\lib\jsynoptic-1.0.jar;%SPISPATHR%\ThirdPart\JSynoptic\lib\jsynoptic-parser-1.0.jar;%SPISPATHR%\ThirdPart\JSynoptic\lib\simtools-1.0.jar;%SPISPATHR%\ThirdPart\JSynoptic\lib\syn3d-1.0.jar
set CLASSPATH=%JSYNOPTICPATH%;%CLASSPATH%

REM #####
REM # Setting of the jModule (Normally do not need any #
REM # modifications. #
REM #####
set JMODULEPATH=%SPISPATHR%\jModule\jModuleUI.jar;%SPISPATHR%\SpisUI\Lib\JFreeMesh\JFreeMesh.jar;%SPISPATHR%\SpisUI\Lib\JFreeMesh\JFreeMesh-VtkBuilder.jar
set CLASSPATH=%JMODULEPATH%;%CLASSPATH%

```

7 Annexe 2: examples of SPIS-UI tracks

7 1. Processing script

```
# Example of Spis Music Track (SMT) script or "How to use SPIS as
# simple Lib through a simple Jython script".
# This script can be directly launched as follow:
#     spis_task.sh -b thisScript.py
# or via the JyConsole.
#
# Author: M.Biais (Artenum), S.Jourdain (Artenum)
# (c) Artenum SARL, Paris, 2005

# to import the SPIS stuff in order to :
# - exchange
# - convert
# - edit
# easily all data and other stuffs
from Bin.GeoManager import GeoManager
from Bin.Tasks.TaskManager import TaskManager
from Bin.Tasks.shared import *
from Bin.Tasks.taskslist import TasksList
import spis

# Initialize Task Manager (if you want to use it)
tasksList = TasksList()
tasksList.initTasksList()
tasksList.setAllAlive()
task_manager = TaskManager(*[i[0] for i in tasksList.tasks])
task_manager.reset_done_nodes()

# To load the project (just by calling the LoadProj Task)
# The project should contain all needed Data. This can
# also be done by direct Python import (see documentation).
project_path = "../Data/ValidationTest/test3_sphere_low_resol/"
sharedFiles["project_directory"] = project_path
sharedFiles["projectLoadingFlag"] = 1 # to notify that the project is already defined

sharedTasks["context"] = []
sharedTasks["context"] = [ "projectInfo",
                           "geomFile",
                           "materials",
                           "elecNodes",
                           "plamas",
                           "groups",
                           "globals",
                           "geomLoading",
                           "meshLoading"]

# here the task manager will perform automatically all pre-processing tasks
# according the dependence tree
task_manager.set_done_task("LoadProj")
tasks_list = ["ProjectLoaderFormat2", "Mesher3D"] # "FieldManager"
# Run tasks declared in the tasklist
for i in tasks_list:
    task_manager.run_tasks(i)
print "The Job is done ! Bye !"
```

7 2. Simulation demon

```
#####
#                               #
#                               #
#####
GL_TRACK_VERSION = 4.3
print "Daemon file: ", daemonFile
print "Daemon path: ", daemonPath
pathProjectIn = daemonPath
pathProjectOut = daemonPath+os.path.basename(daemonFile)[23:-3]
##### END OF HEADER #####

GL_TRACK_VERSION = 4.3

print "Daemon file: ", daemonFile
print "Daemon path: ", daemonPath
pathProjectIn = daemonPath
pathProjectOut = daemonPath+os.path.basename(daemonFile)[23:-3]

import spis
from Bin.Tasks.shared import *
from Bin.ProjectLoader2 import ProjectLoader2
from Bin.SpisNumInterface import *
from Bin.JyTop4 import JyTop4
from Modules.SolverInterfaces.SpisNum.SpisNumSimulationDataExtractor \
    import SpisNumSimulationDataExtractor
from Bin.ProjectWriter2 import ProjectWriter2
from Bin.config import GL_SPIS_VERSION

print "The long track begins..."
# The project should contain all needed Data. This can
# also be done by direct Python import (see documentation).
sharedFiles["project_directory"] = pathProjectIn
sharedFiles["projectLoadingFlag"] = 1 # to notify that the project is already defined
# we load only the needed elements, i.e project s general info, global parameters
# preprocessing DataFields and MeshFields, the mesh structure.
loadingList=["projectInfo","globals","preproDFLoading","meshLoading","numKernelParam",\
    "materials"]
loader = ProjectLoader2()
loader.setLoadingList(loadingList)
loader.load()

Interface = SpisNumInterface(sharedData["AllDataField"], sharedData["AllMeshField"])
Interface.BuildList(shared['Mesh'])
Interface.MapDFOnVolMesh()
Interface.MapDFOnSCMesh()
# xxxx
Interface.applyMaterialProperties(sharedProp['materialModel'],\
    sharedProp['selectedNascapMaterialList'])

Interface.MapDFOnBdMesh()
Interface.buildCrossNumberingBetweenMesh()
sharedNum['SNMesh'] = Interface.GetSpisNumMesh()
print "          DONE"
print ""
print " ##### "
print ""
print "Starting the simulation"
print ""
print " ##### "
```



```

print "materialModel= ", sharedProp['materialModel']
if (sharedProp['materialModel'] == None): sharedProp['materialModel'] = 'legacy'

print "selectedNascapMaterialList= ",sharedProp['selectedNascapMaterialList']

sharedSolver["jytop"] = JyTop4(sharedNum["SNMesh"], sharedData["AllDataField"],\
                               sharedData["AllMeshField"])
if sharedProp['selectedNascapMaterialList']!=None:\
    sharedSolver["jytop"].buildNascapParameterSetList(sharedProp['materialModel'],\
                                                       sharedProp['selectedNascapMaterialList'],\
                                                       sharedProp['defaultNascapMaterialList'])
if sharedProp['selectedNascapMaterialList'] == None:\
    sharedSolver["jytop"].setSharedMaterialModel(sharedProp['materialModel'])

sharedSolver["jytop"].setGlobalParameters(sharedGlobals)
sharedSolver["jytop"].BuildSim()
sharedSolver["jytop"].Run()

print " ##### "
print " Data mining "
print " ##### "
#sharedSolver["jytop"].ReadSim( sharedData["AllMeshField"], sharedData["AllDataField"])
extractor = SpisNumSimulationDataExtractor()
extractor.setInput(sharedSolver["jytop"].simu)
extractor.setOutputDataBus( sharedData["AllMeshField"], sharedData["AllDataField"])

extractor.setDefaultSimulationId()
print "*****"
extractor.readVolumeData()
print "*****"
extractor.readSCSSurfaceData()
print "*****"
extractor.readBoundarySurfaceData()
print "*****"
extractor.readTimeDependentData()
print "*****"
extractor.readParticleTrajectories()
print "*****  END  *****"

print "Data recovering"
writer = ProjectWriter2()
writer.setOutputDirectory(pathProjectOut)
writer.createNewProject()
writer.write()

print "The Job is done ! Bye !"

```

8 Annexe 3: JyTop4.py script

```

import os, time
import java
from spis.Top.Simulation import *
from spis.Top.SC import *
from spis.Top.Plasma import *
from spis.Vol.VolMesh import *
from spis.Vol.VolField import *
from spis.Vol.VolDistrib import *
from spis.Vol.Geom import *
from spis.Surf.SurfMesh import *
from spis.Surf.SurfField import *
from spis.Util.Table import *
from spis.Util.Monitor import *
from spis.Top.Top import *
import spis.Top.Default.LocalParameter as LocalParameter
from Modules.Field.DataField import DataField
from Modules.Field.DataFieldList import DataFieldList
from Modules.Field.MeshField import MeshField
from Modules.Field.MeshFieldList import MeshFieldList
from math import sqrt
from config import GL_EXCHANGE, GL_DEFAULT_INPUT_PATH, GL_DATA_PATH
from Bin.Tasks.shared import sharedFiles
from Bin.Tasks.shared import sharedData
from Bin.Tasks.shared import sharedData

class JyTop4:
    '''
    Top simulation kernel wrapper. This Jython class wrap the
    java based SPIS-NUM simulation kernel. Its performs the data
    structure conversion from UI to NUM, the simulation model
    building, launches the execution of the simulation loop and
    recovers the output data.
    '''

    def __init__(self, spisNumMesh):
        tmpDate = time.gmtime()
        self.simulationId = `tmpDate[0]`+`tmpDate[1]`+`tmpDate[2]`+`tmpDate[3]`+`tmpDate[4]`
+`tmpDate[5]`
        print "Initialisation of simulation Nb:", self.simulationId

        self.logFile = os.path.join(GL_EXCHANGE, "spis_JyTop4.log")
        self.stream = open(self.logFile, "w")

        self.logFile2 = os.path.join(GL_EXCHANGE, "spis_JyTop4-2.log")
        self.stream2 = open(self.logFile2, "w")
        self.spisNumMesh = spisNumMesh

    #####
    # Create and prepare the simulation #
    #####
    def BuildSim(self, AllMeshField, AllDataField):
        '''
        Create and prepare the grids and the simulation model.
        '''
        print "SPIS/NUM main is under creation"

        # GEOMETRY INITIALISATION SECTION, MIMICKING (SEMI-)AUTOMATED
        # DATA TRANSFER FROM FRAMEWORK TO SPIS NUM

        # Creates meshes and some property fields from framework data.
        # (Of course these definitions and initialisations shall be
        # different in the framework, they are just written here for
        # testing).
        #

```

```

# Data are:
# - VolMesh (see ThreeDUnstructVolMesh for more on
#           variable meaning and some classical
#           constraints on numbering) :
# cell (tetrahedra) number
self.geom = ThreeDCartesianGeom()

print >> self.stream, "creation of the ThreeDUnstructSurfMesh structure (for SC surface)"

# -----
# - Init for spacecraft -
# -----
self.scSurfMesh = \
    ThreeDUnstructSurfMesh(self.geom,
        None,
        self.spisNumMesh.getSpacecraftMesh().getFaceNb(),
        self.spisNumMesh.getSpacecraftMesh().getEdgeNb(),
        self.spisNumMesh.getSpacecraftMesh().getNodeNb(),
        self.spisNumMesh.getSpacecraftMesh().getSurfEdge(),
        self.spisNumMesh.getSpacecraftMesh().getSurfNode(),
        self.spisNumMesh.getSpacecraftMesh().getEdgeNode(),
        self.spisNumMesh.getSpacecraftMesh().getData('SurfFlagS'),
        self.spisNumMesh.getSpacecraftMesh().getSurfIndex(),
        [],
        [0 for i in range(self.spisNumMesh.getSpacecraftMesh().getFaceNb())],
        self.spisNumMesh.getSpacecraftMesh().getData('EdgeFlagS'),
        self.spisNumMesh.getSpacecraftMesh().getEdgeIndex(),
        [],
        [0 for i in range(self.spisNumMesh.getSpacecraftMesh().getEdgeNb())],
        self.spisNumMesh.getSpacecraftMesh().getData('NodeFlagS'),
        self.spisNumMesh.getSpacecraftMesh().getNodeIndex(),
        [],
        self.spisNumMesh.getSpacecraftMesh().getXYZ())

# -----
# - Init for boundary -
# -----

print >> self.stream, "creation of the ThreeDUnstructSurfMesh structure (for external
boudnary surface)"
self.bdSurfMesh = \
    ThreeDUnstructSurfMesh(self.geom,
        None,
        self.spisNumMesh.getBoundaryMesh().getFaceNb(),
        self.spisNumMesh.getBoundaryMesh().getEdgeNb(),
        self.spisNumMesh.getBoundaryMesh().getNodeNb(),
        self.spisNumMesh.getBoundaryMesh().getSurfEdge(),
        self.spisNumMesh.getBoundaryMesh().getSurfNode(),
        self.spisNumMesh.getBoundaryMesh().getEdgeNode(),
        self.spisNumMesh.getBoundaryMesh().getData('SurfFlagBd'),
        self.spisNumMesh.getBoundaryMesh().getSurfIndex(),
        [-1 for i in range(self.spisNumMesh.getBoundaryMesh().getFaceNb())],
        [0 for i in range(self.spisNumMesh.getBoundaryMesh().getFaceNb())],
        self.spisNumMesh.getBoundaryMesh().getData('EdgeFlagBd'),
        self.spisNumMesh.getBoundaryMesh().getEdgeIndex(),
        [-1 for i in range(self.spisNumMesh.getBoundaryMesh().getEdgeNb())],
        [0 for i in range(self.spisNumMesh.getBoundaryMesh().getEdgeNb())],
        self.spisNumMesh.getBoundaryMesh().getData('NodeFlagBd'),
        self.spisNumMesh.getBoundaryMesh().getNodeIndex(),
        [],
        self.spisNumMesh.getBoundaryMesh().getXYZ())

# -----
# - Init for volume -
# -----
print >> self.stream, "creation of the ThreeDUnstructVolMesh structure (for Plasma Volume)"

```

```

print "creation of the ThreeDUnstructVolMesh structure (for Plasma Volume)"

self.volMesh = \
    ThreeDUnstructVolMesh(self.geom
        self.bdSurfMesh
        self.scSurfMesh
        self.spisNumMesh.getVolumeMesh().getCellNb()
        self.spisNumMesh.getVolumeMesh().getFaceNb()
        self.spisNumMesh.getVolumeMesh().getEdgeNb()
        self.spisNumMesh.getVolumeMesh().getNodeNb()
        self.spisNumMesh.getVolumeMesh().getCellSurf()
        self.spisNumMesh.getVolumeMesh().getCellEdge()
        self.spisNumMesh.getVolumeMesh().getCellNode()
        self.spisNumMesh.getVolumeMesh().getSurfEdge()
        self.spisNumMesh.getVolumeMesh().getSurfNode()
        self.spisNumMesh.getVolumeMesh().getEdgeNode()
        self.spisNumMesh.getVolumeMesh().getData('SurfFlag')
        self.spisNumMesh.getVolumeMesh().getSurfIndexSC()
        []
        self.spisNumMesh.getVolumeMesh().getSurfIndexB()
        self.spisNumMesh.getVolumeMesh().getData('EdgeFlag')
        self.spisNumMesh.getVolumeMesh().getEdgeIndexSC()
        []
        self.spisNumMesh.getVolumeMesh().getEdgeIndexB()
        self.spisNumMesh.getVolumeMesh().getData('NodeFlag')
        self.spisNumMesh.getVolumeMesh().getNodeIndexSC()
        []
        self.spisNumMesh.getVolumeMesh().getNodeIndexB()
        self.spisNumMesh.getVolumeMesh().getXYZ()
    )

# -----
# - Linking surface meshes to volume mesh -
# -----
print >> self.stream, "Linking surface meshes to volume mesh"
print "Linking surface meshes to volume mesh"

print "STEP 0"
self.bdSurfMesh.setVm(self.volMesh)

print "STEP 1"
self.scSurfMesh.setVm(self.volMesh)

# -----
# - Global Param -
# -----
print >> self.stream, "creating Global Parameter data"
print "creating Global Parameter data"
from spis.Top.Simulation import *
from spis.Top.Default import *
from Bin.Tasks.shared import sharedGlobals
self.globalParameterArray = []
for key in sharedGlobals.keys():
    p = sharedGlobals[key]
    param = GlobalParameter(key,p[2],p[4],p[3],p[1])
    self.globalParameterArray.append(param)

# -----
# - Local Param -
# -----
self.localParameterArray = []

# -----
# - Local Param for Volume -
# -----
print "##### Mapping of Data in Volume #####"
paramNameList = [ ['VolInteracFlag', 3] ,\
    ['BackGroundDens', 3] ]

```

```

spisMesh      = self.spisNumMesh.getVolumeMesh()
spisNumMesh   = self.volMesh
self.buildLocalParamFromList(paramNameList, spisMesh, spisNumMesh)

# -----
# - Local Param for Spacecraft -
# -----
print "##### Mapping of Data on S/C surface #####"
paramNameList = [ ['MatModelId'      , 2] , \
                  ['MatTypeId'       , 2] , \
                  ['MatThickness'     , 2] , \
                  ['PhotoEmis'       , 2] , \
                  ['ElecSecEmis'     , 2] , \
                  ['ProtSecEmis'     , 2] , \
                  ['VolConduct'      , 2] , \
                  ['IndConduct'      , 2] , \
                  ['SurfConduct'     , 2] , \
                  ['Temperature'     , 2] , \
                  ['SunFlux'         , 2] , \
                  ['ElecNodeId'      , 2] , \
                  ['EdgeElecNodeId'  , 1] , \
                  ['SCDiriFlag'     , 0] , \
                  ['SCDiriPot'      , 0] , \
                  ['SCDiriPotEdge'   , 1] , \
                  ['SCDiriPotSurf'   , 2] , \
                  ['SCFourFlag'     , 2] , \
                  ['SCFourAlpha'    , 2] , \
                  ['SCFourValue'    , 0] , \
                  ['SourceId'       , 2] , \
                  ['SourceCurrent'   , 2] , \
                  ['SourceTemp'     , 2] , \
                  ['SourceMach'     , 2] ]

spisMesh      = self.spisNumMesh.getSpacecraftMesh()
spisNumMesh   = self.scSurfMesh
self.buildLocalParamFromList(paramNameList, spisMesh, spisNumMesh)

# -----
# - Local Param for Boundary -
# -----
print "##### Mapping of Data on External BD #####"
paramNameList = [ ['BdDiriFlag'     , 0] , \
                  ['BdDiriPot'     , 0] , \
                  ['BdFourFlag'    , 2] , \
                  ['BdFourAlpha'   , 2] , \
                  ['BdFourValue'   , 0] , \
                  ['IncomPart'     , 2] , \
                  ['OutgoPart'     , 2] ]

spisMesh      = self.spisNumMesh.getBoundaryMesh()
spisNumMesh   = self.bdSurfMesh
self.buildLocalParamFromList(paramNameList, spisMesh, spisNumMesh)

self.runId = -1
print "Now, everything should be ready for the simulation"

#####
#          CALLING SPIS-NUM:          #
# Major place where modifications can be done by users #
#####
print "GL_DATA_PATH:", GL_DATA_PATH
print "GL_DEFAULT_INPUT_PATH: ", GL_DEFAULT_INPUT_PATH
import os
os.listdir(GL_DEFAULT_INPUT_PATH)

print "Building of the simulation model"
print ">> self.stream, \"creating GEO simulation java objet in SPIS/NUM\"
self.simu = SimulationFromUIParams( self.volMesh,
```

```

        self.bdSurfMesh,
        self.scSurfMesh,
        self.globalParameterArray,
        self.localParameterArray,
        GL_DATA_PATH,
        GL_DEFAULT_INPUT_PATH)

def buildLocalParamFromList(self, list, spisMesh, spisNumMesh):
    for param in list:
        self.buildLocalParam(self.localParameterArray, param[0], param[1], spisMesh, spisNumMesh)

def buildLocalParam(self, paramArray, localName, localisation, spisMesh, spisNumMesh):

    paramArray.append(LocalParameter( localName,
                                       spisMesh.convertDataToFloatArray(localName),
                                       localisation, spisNumMesh,
                                       sharedData["AllDataField"].Dic[localName].Unit,
                                       'no comment'))

def Run(self):
    print "SPIS/NUM main is starting"
    self.runId = self.runId + 1
    print "Run Nb:", self.runId
    print >> self.stream, "Integration in SPIS/NUM"
    self.simu.integrate()
    self.simu.close()
    print >> self.stream, "Back from SPIS/NUM"

#####      Examples of possible modifications:      #####
# GEO simulation: change the above 5 lines into the following #
# (uncommented them! and comment the above ones):           #

#print >> self.stream, "creating GEO simulation java objet in SPIS/NUM"
#self.simu = GeoExample2(self.volMesh, self.bdSurfMesh,
#self.scSurfMesh,self.globalParameterArray)
#print >> self.stream, "Integration in SPIS/NUM"
#self.simu.integrate()
#print >> self.stream, "Back from SPIS/NUM"

# LEO simulation: change the above 5 lines into the following #
# (uncommented them! and comment the above ones):           #

#print >> self.stream, "creating LEO simulation java objet in SPIS/NUM"
#self.simu = LeoExample(self.volMesh, self.bdSurfMesh, self.scSurfMesh)
#print >> self.stream, "Integration in SPIS/NUM"
#self.simu.integrate(0.001)
#print >> self.stream, "Back from SPIS/NUM"

# GEO simulation with photo-emission turned on after a while: #
#print >> self.stream, "creating GEO simulation java objet in SPIS/NUM (with photo-
#emission)"
#self.simu = GeoExample2(self.volMesh, self.bdSurfMesh, self.scSurfMesh)
#print >> self.stream, "Integration in SPIS/NUM"
#self.simu.integrate(1.0)
#print >> self.stream, "Back from SPIS/NUM"

# To dump meshes, and be able to run SPIS-NUM as a standalone #
# code (within Eclipse), and reload these meshes through the #
# import command in the SPIS-NUM menu used when standalone:   #

def ReadSim(self, AllMeshField, AllDataField):
    print "SPIS/NUM: Reading of the results"

#####
# Section of return of output data from num to ui           #
#####
print >> self.stream, " "
print >> self.stream, "Getting of output data

```

```
#####
# S/C data                                     #
#####
print "Getting now the spacecraft surface data for postprocessing"
print "\n"
self.scSurfResults = self.simu.getScSurfResults()
# self.scSurfResults should now contain a SurfField[]
print >> self.stream, \
'back from NUM: SC surface fields = ',self.scSurfResults

self.dataIndex = max(AllDataField.IdList)
print "Data Index start Value ", self.dataIndex
for i in range(len(self.scSurfResults)):
    if self.scSurfResults[i] != None:          #we check if the object exist
        #print self.scSurfResults[i].getName()
        self.dataIndex = self.dataIndex + 1
        surfData = DataField()
        surfData.Id = self.dataIndex
        surfData.Name = self.scSurfResults[i].getName()
        surfData.Type = 'Float (TBC)'
        surfData.Unit = self.scSurfResults[i].getUnit().getLongName()
        surfData.Local = self.scSurfResults[i].getCentring().getAbsDim()
        surfData.LockedValue = 0
        tmpValueList = self.scSurfResults[i].getTable().getValues()
        surfData.ValueList = []
        for elm in tmpValueList:
            surfData.ValueList.append(elm)
        surfData.Category = "outputSim"+self.simulationId+"Run"+`self.runId`+"DataOnSC"

        if surfData.Local == 0:
            surfData.MeshFieldId = AllMeshField.Dic['NodeFlagS_MF'].Id
            if len(AllMeshField.Dic['NodeFlagS_MF'].MeshElementIdList) != len
(surfData.ValueList):
                print "ERROR datafield vs meshfield, SC ", i, \
                    len(AllMeshField.Dic['NodeFlagS_MF'].MeshElementIdList),\
                    len(surfData.ValueList), surfData.Local
            if surfData.Local == 1:
                surfData.MeshFieldId = AllMeshField.Dic['EdgeFlagS_MF'].Id
                if len(AllMeshField.Dic['EdgeFlagS_MF'].MeshElementIdList) != len
(surfData.ValueList):
                    print "ERROR datafield vs meshfield, SC ", i, \
                        len(AllMeshField.Dic['EdgeFlagS_MF'].MeshElementIdList),\
                        len(surfData.ValueList), surfData.Local
            if surfData.Local == 2:
                surfData.MeshFieldId = AllMeshField.Dic['SurfFlagS_MF'].Id
                if len(AllMeshField.Dic['SurfFlagS_MF'].MeshElementIdList) != len
(surfData.ValueList):
                    print "ERROR datafield vs meshfield, SC ", i, \
                        len(AllMeshField.Dic['SurfFlagS_MF'].MeshElementIdList),\
                        len(surfData.ValueList), surfData.Local
        AllDataField.Add_DataField(surfData)
        print surfData.Name
        print >> self.stream, surfData.Name
        print >> self.stream, surfData.ValueList
        print >> self.stream, "\n"

#####
# Boundary surface data                       #
#####
print "\n"
print "Getting now the boundary surface data for posprocessing"
print "\n"
self.bdSurfResults = self.simu.getBdSurfResults()
# self.bdSurfResults should now contain a SurfField[]
print >> self.stream, \
'back from NUM: boundary surface fields = ',self.bdSurfResults
```

```

for i in range(len(self.bdSurfResults)):
    if self.bdSurfResults[i] != None:          #we check if the object exist
        #print self.bdSurfResults[i].getName()
        bdsurfData = DataField()
        self.dataIndex = self.dataIndex + 1
        bdsurfData.Id = self.dataIndex
        bdsurfData.Name = self.bdSurfResults[i].getName()
        bdsurfData.Type = 'Float (TBC)'
        bdsurfData.Unit = self.bdSurfResults[i].getUnit().getLongName()
        bdsurfData.Local = self.bdSurfResults[i].getCentring().getAbsDim()
        bdsurfData.LockedValue = 0 #TBC
        #bdsurfData.ValueList = self.bdSurfResults[i].getTable().getValues()
        tmpValueList = self.bdSurfResults[i].getTable().getValues()
        bdsurfData.ValueList = []
        for elm in tmpValueList:
            bdsurfData.ValueList.append(elm)
        bdsurfData.Category = "outputSimu"+self.simulationId+"Run"+`self.runId`+"DataOnBd"
        bdsurfData.MeshFieldId = AllMeshField.Dic['NodeFlagBd_MF'].Id
        if bdsurfData.Local == 0:
            bdsurfData.MeshFieldId = AllMeshField.Dic['NodeFlagBd_MF'].Id
            if len(AllMeshField.Dic['NodeFlagBd_MF'].MeshElementIdList) != len
(bdsurfData.ValueList):
                print "ERROR datafield vs meshfield, Bd ", i, \
                    len(AllMeshField.Dic['NodeFlagBd_MF'].MeshElementIdList), \
                    len(bdsurfData.ValueList), bdsurfData.Local
            if bdsurfData.Local == 1:
                bdsurfData.MeshFieldId = AllMeshField.Dic['EdgeFlagBd_MF'].Id
                if len(AllMeshField.Dic['EdgeFlagBd_MF'].MeshElementIdList) != len
(bdsurfData.ValueList):
                    print "ERROR datafield vs meshfield, Bd ", i, \
                        len(AllMeshField.Dic['EdgeFlagBd_MF'].MeshElementIdList), \
                        len(bdsurfData.ValueList), bdsurfData.Local
            if bdsurfData.Local == 2:
                bdsurfData.MeshFieldId = AllMeshField.Dic['SurfFlagBd_MF'].Id
                if len(AllMeshField.Dic['SurfFlagBd_MF'].MeshElementIdList) != len
(bdsurfData.ValueList):
                    print "ERROR datafield vs meshfield, Bd ", i, \
                        len(AllMeshField.Dic['SurfFlagBd_MF'].MeshElementIdList), \
                        len(bdsurfData.ValueList), bdsurfData.Local
            # XXX see SC section for extension to properties living on edge and surf XXX
            AllDataField.Add_DataField(bdsurfData)
        print bdsurfData.Name
        print >> self.stream, bdsurfData.Name
        print >> self.stream, bdsurfData.ValueList
        print >> self.stream, "\n"

#####
# Volumic data #
#####
print "\n"
print "Getting now the volume data for posprocessing"
print "\n"
self.volResults = self.simu.getVolResults()
# self.volResults should now contain a VolField[]
# use .buildExternIndexedTable().getValues() on each of the volField
print >> self.stream, \
'plasma potential back from NUM', self.volResults

for i in range(len(self.volResults)):
    if self.volResults[i] != None:          #we check if the object exist
        #print self.volResults[i].getName()
        self.volDat = DataField()
        self.dataIndex = self.dataIndex + 1
        self.volDat.Id = self.dataIndex
        self.volDat.Name = self.volResults[i].getName()
        self.volDat.Type = 'Float TBC'
        self.volDat.Unit = self.volResults[i].getUnit().getLongName()

```



```

        self.volDat.Local = 0
        self.volDat.LockedValue = 0
        #self.volDat.ValueList = self.volResults[i].buildExternIndexedTable().getValues()
        tmpValueList = self.volResults[i].buildExternIndexedTable().getValues()
        self.volDat.ValueList = []
        for elm in tmpValueList:
            self.volDat.ValueList.append(elm)
        self.volDat.Category = "outputSimu"+self.simulationId+"Run"+`self.runId`
+"DataInvolve"
        self.volDat.MeshFieldId = AllMeshField.Dic['NodeFlag_MF'].Id
        # XXX see SC section for extension to properties living on edge and surf XXX
        if len(AllMeshField.Dic['NodeFlag_MF'].MeshElementIdList) != len
(self.volDat.ValueList):
            print "ERROR datafield vs meshfield, Vol",i\
                ,len(AllMeshField.Dic['NodeFlag_MF'].MeshElementIdList),
                len(self.volDat.ValueList)
        AllDataField.Add_DataField(self.volDat)
        print self.volDat.Name
        print >> self.stream, self.volDat.Name
        print >> self.stream, self.volDat.ValueList
        print >> self.stream, "\n"

#####
# Time dependent data                                     #
#####
print >> self.stream, \
    "Getting now the x(t) data"
print "Getting now the x(t) data"
fieldIndex = max(AllMeshField.IdList)
fieldIndex = fieldIndex + 1
print "New MeshFields start at:", fieldIndex

self.scalTimePlots = self.simu.getScalTimePlots()
# self.scalTimePlots should now contain a XyData[]
print >> self.stream, \
    'back from NUM: x(t) data = ',self.scalTimePlots
sharedData["scalTimePlots"] = self.scalTimePlots

for i in range(len(self.scalTimePlots)):
    if self.scalTimePlots[i] != None:          #we check if the object exist
        nbValideData = self.scalTimePlots[i].getDataNb()
        #first, we build the MesField corresponding to the time.
        #for now, time is expressed in number of time step only.
        # for the moement, we build a MeshField for each time-plot and we store directly the
        # the time. But this should be chenged into a "time-mesh" in the future
        #self.indexedTime = [i for i in range(len(self.scalTimePlots[0].YArray))]
        self.timeMeshField = MeshField()
        self.timeMeshField.Id = fieldIndex
        self.timeMeshField.Name = 'Time'
        self.timeMeshField.Local = 4
        self.timeMeshField.Description = "time in steps"

        tmpArray = range(nbValideData)
        for j in xrange(nbValideData):
            tmpArray[j] = self.scalTimePlots[i].XArray[j]

        self.timeMeshField.MeshElementList = tmpArray
        self.timeMeshField.MeshElementIdList = tmpArray    #this is not clean but...

        AllMeshField.Add_MeshField(self.timeMeshField)

        self.timeDat = DataField()
        tmpArray = range(nbValideData)
        tmpIdArray = range(nbValideData)
        for j in xrange(nbValideData):
            tmpArray[j] = self.scalTimePlots[i].YArray[j]
            tmpIdArray[j] = j

```

```

        self.dataIndex = self.dataIndex + 1
        self.timeDat.Id = self.dataIndex
        self.timeDat.Name = self.scalTimePlots[i].getName()
        self.timeDat.Type = "FLOAT" #tmpArray[1]
        self.timeDat.Unit = self.scalTimePlots[i].getYTable().getUnit().getLongName()
        self.timeDat.Local = 4 #time is a curviline absciss
        self.timeDat.LockedValue = 0
        self.timeDat.ValueList = tmpArray
        self.timeDat.MeshElementIdList = tmpIdArray
        self.timeDat.MeshFieldId = fieldIndex
        self.timeDat.Category = "outputSimu"+self.simulationId+"Run"+`self.runId`
+"TimeDependentData"
        AllDataField.Add_DataField(self.timeDat)
        print self.timeDat.Name
        print >> self.stream, self.timeDat.Name
        print >> self.stream, self.timeDat.ValueList
        print >> self.stream, "\n"
        fieldIndex = fieldIndex + 1

print >> self.stream, \
"Getting now the (x(y))(t) data"

self.curveTimePlots = self.simu.getCurveTimePlots()
# self.curveTimePlots should now contain a XyzData[]
print >> self.stream, \
'back from NUM: (x(y))(t) data = ',self.curveTimePlots

#####
# Particle trajectories #
#####
print >> self.stream, \
"Getting now the trajectory data"
print "Getting now the trajectory data"

self.trajectories = self.simu.getTrajectories()
# self.trajectories should now contain a Trajectory[]
fieldIndex = max(AllMeshField.IdList)
fieldIndex = fieldIndex + 1
print "New MeshFields start at:", fieldIndex

sharedData["particleTrajectories"] = self.trajectories

for i in range(len(self.trajectories)):
    if self.trajectories[i] != None: #we check if the object exist

        nbPoints = self.trajectories[i].getPointNb()
        #first, we build the MesField corresponding to the traj.
        #self.indexedTime = [i for i in range(len(self.scalTimePlots[0].YArray))]
        self.trajMeshField = MeshField()
        self.trajMeshField.Id = fieldIndex
        self.trajMeshField.Name = 'trajectory'
        self.trajMeshField.Local = 5
        self.trajMeshField.Description = "coordinates"

        #tmpArray = self.scalTimePlots[i].XArray
        tmpArray = range(nbPoints)
        for j in xrange(nbPoints):
            tmpArray[j] = (self.trajectories[i].getPosArray()[j][0], self.trajectories
[i].getPosArray()[j][1], self.trajectories[i].getPosArray()[j][2])

        self.trajMeshField.MeshElementList = tmpArray #self.indexedTime
        AllMeshField.Add_MeshField(self.trajMeshField)

        self.velocity = DataField()

        #tmpArray = self.scalTimePlots[i].YArray

```

```

        tmpArray = range(nbPoints)
        tmpIdArray = range(nbPoints)
        for j in xrange(nbPoints):
            tmpArray[j] = (self.trajectories[i].getPosArray()[j][0], self.trajectories
[i].getPosArray()[j][1], self.trajectories[i].getPosArray()[j][2])
            tmpIdArray[j] = j
            self.dataIndex = self.dataIndex + 1
            self.velocity.Id = self.dataIndex
            self.velocity.Name = self.trajectories[i].getName()
            self.velocity.Type = "FLOAT" #tmpArray[1]
            self.velocity.Unit = 'see name'
            self.velocity.Local = 5
            self.velocity.LockedValue = 0
            self.velocity.ValueList = tmpArray
            self.velocity.MeshElementIdList = tmpIdArray
            self.velocity.MeshFieldId = fieldIndex
            self.velocity.Category = "outputSimu"+self.simulationId+"Run"+`self.runId`
+"ParticuleTrajectory"
            # XXX see SC section for extension to properties living on edge and surf XXX
            #if len(AllMeshField.Dic['NodeFlag_MF'].MeshElementIdList) != len
(self.volDat.ValueList):
            #    print "ERROR datafield vs meshfield, Vol",i\
            #    ,len(AllMeshField.Dic['NodeFlag_MF'].MeshElementIdList),len
(self.volDat.ValueList)
            AllDataField.Add_DataField(self.velocity)
            print self.velocity.Name
            print >> self.stream, self.velocity.Name
            print >> self.stream, self.velocity.ValueList
            print >> self.stream, "\n"
            fieldIndex = fieldIndex + 1

print >> self.stream, \
'back from NUM: trajectory data = ',self.trajectories

##### END OF THE DATA EXTRACTION #####

# more generally for a field f (a SurfField or VolField, as
# self.top.getScPotential() here):
# - f.getTable().getValues() returns the values (float[]
#   or float[][] for a VectxxxField)
# - for node-centred volume fields, which have a different
#   internal indexing, one must use
#   f.buildExternIndexedTable().getValues() (f.getTable().getValues() returns values
#   - the centring of the field can be tested through
#     self.top.getScPotential().getCentring().getAbsDim(),
#     which returns 0 for nodes, 2 for surfaces...
# - the mesh this field lives on is obtained by
#   - f.getSm() for a SurfField (must be either
#     self.bdSurfMesh or self.scSurfMesh)
#   - f.getVm() for a VolField (must be self.volMesh)
# NB: a generic method f.getMesh() might be written in java,
#     if good for jython
# NB2: the meshes are nor duplicated in java, hence testing
#       whether f.getSm() is self.bdSurfMesh or self.scSurfMesh,
#       can in principle be done by comparing references

```

9 Annexe 4: Task dependence tree, tasklist.py script.

```
import sys

print "Please wait..."
sys.stdout.write("Tasks loading")
from Bin.Tasks.TaskCADImporter
from Bin.Tasks.TaskMesher
from Bin.Tasks.TaskMaterial
from Bin.Tasks.TaskGroupManager
from Bin.Tasks.TaskFieldManager
from Bin.Tasks.TaskSpisNumInterface
from Bin.Tasks.TaskViewPipeline1
from Bin.Tasks.TaskViewPipeline2
from Bin.Tasks.TaskToolCaller
from Bin.Tasks.TaskEditIni
from Bin.Tasks.TaskPrompt
from Bin.Tasks.TaskFileChooser
from Bin.Tasks.TaskJyTop
from Bin.Tasks.TaskMesher2D
from Bin.Tasks.TaskMesher3D
from Bin.Tasks.TaskBuildDataFieldChooser
from Bin.Tasks.TaskBuildPlot2D
from Bin.Tasks.TaskEditNumSettings
from Bin.Tasks.TaskGroupEditor
from Bin.Tasks.TaskReloadSolver
from Bin.Tasks.TaskSaveProj
from Bin.Tasks.TaskSaveProjAs
from Bin.Tasks.TaskLoadProj
from Bin.Tasks.TaskMaterialEditor
from Bin.Tasks.TaskElecNodeEditor
from Bin.Tasks.TaskPlasmaEditor
from Bin.Tasks.TaskDataEditor
from Bin.Tasks.TaskReload
from Bin.Tasks.TaskInitGroup
from Bin.Tasks.TaskConvertGroup
from Bin.Tasks.TaskSaveCAD
from Bin.Tasks.TaskLoadCAD
from Bin.Tasks.TaskSaveProperties
from Bin.Tasks.TaskLoadProperties
from Bin.Tasks.TaskNewProject
from Bin.Tasks.TaskSolverInit
from Bin.Tasks.TaskSolverBuildSim
from Bin.Tasks.TaskSolverRun
from Bin.Tasks.TaskSolverReadSim
from Bin.Tasks.TaskParaview
from Bin.Tasks.TaskParamsEditor
from Bin.Tasks.TaskSaveGroups
from Bin.Tasks.TaskLoadGroups
from Bin.Tasks.TaskDocCaller
from Bin.Tasks.TaskCallCassandra
from Bin.Tasks.TaskCallJyConsole
from Bin.Tasks.TaskCallMemoryMonitor
from Bin.Tasks.TaskCallJSynoptic
from Bin.Tasks.TaskMeshSplitter
from Bin.Tasks.TaskInitFields
from Bin.Tasks.TaskExit
from Bin.Tasks.TaskExportToGmshMesh
from Bin.Tasks.TaskMeshImporter
from Bin.Tasks.TaskDataBusCleaner
from Bin.Tasks.TaskCallNumParamEditor
from Bin.Tasks.TaskEditCharScales
from Bin.Tasks.TaskPicUpParamsEditor
from Bin.Tasks.TaskExportToPicUp
from Bin.Tasks.TaskCallPicUp3D
from Bin.Tasks.TaskCallPicUp3DReload
from Bin.Tasks.TaskProjectControler
from Bin.Tasks.TaskProjectLoaderFormat2

import TaskCADImporter; sys.stdout.write(".")
import TaskMesher; sys.stdout.write(".")
import TaskMaterial; sys.stdout.write(".")
import TaskGroupManager; sys.stdout.write(".")
import TaskFieldManager; sys.stdout.write(".")
import TaskSpisNumInterface; sys.stdout.write(".")
import TaskViewPipeline1; sys.stdout.write(".")
import TaskViewPipeline2; sys.stdout.write(".")
import TaskToolCaller; sys.stdout.write(".")
import TaskEditIni; sys.stdout.write(".")
import TaskPrompt; sys.stdout.write(".")
import TaskFileChooser; sys.stdout.write(".")
import TaskJyTop; sys.stdout.write(".")
import TaskMesher2D; sys.stdout.write(".")
import TaskMesher3D; sys.stdout.write(".")
import TaskBuildDataFieldChooser; sys.stdout.write(".")
import TaskBuildPlot2D; sys.stdout.write(".")
import TaskEditNumSettings; sys.stdout.write(".")
import TaskGroupEditor; sys.stdout.write(".")
import TaskReloadSolver; sys.stdout.write(".")
import TaskSaveProj; sys.stdout.write(".")
import TaskSaveProjAs; sys.stdout.write(".")
import TaskLoadProj; sys.stdout.write(".")
import TaskMaterialEditor; sys.stdout.write(".")
import TaskElecNodeEditor; sys.stdout.write(".")
import TaskPlasmaEditor; sys.stdout.write(".")
import TaskDataEditor; sys.stdout.write(".")
import TaskReload; sys.stdout.write(".")
import TaskInitGroup; sys.stdout.write(".")
import TaskConvertGroup; sys.stdout.write(".")
import TaskSaveCAD; sys.stdout.write(".")
import TaskLoadCAD; sys.stdout.write(".")
import TaskSaveProperties; sys.stdout.write(".")
import TaskLoadProperties; sys.stdout.write(".")
import TaskNewProject; sys.stdout.write(".")
import TaskSolverInit; sys.stdout.write(".")
import TaskSolverBuildSim; sys.stdout.write(".")
import TaskSolverRun; sys.stdout.write(".")
import TaskSolverReadSim; sys.stdout.write(".")
import TaskParaview; sys.stdout.write(".")
import TaskParamsEditor; sys.stdout.write(".")
import TaskSaveGroups; sys.stdout.write(".")
import TaskLoadGroups; sys.stdout.write(".")
import TaskDocCaller; sys.stdout.write(".")
import TaskCallCassandra; sys.stdout.write(".")
import TaskCallJyConsole; sys.stdout.write(".")
import TaskCallMemoryMonitor; sys.stdout.write(".")
import TaskCallJSynoptic; sys.stdout.write(".")
import TaskMeshSplitter; sys.stdout.write(".")
import TaskInitFields; sys.stdout.write(".")
import TaskExit; sys.stdout.write(".")
import TaskExportToGmshMesh; sys.stdout.write(".")
import TaskMeshImporter; sys.stdout.write(".")
import TaskDataBusCleaner; sys.stdout.write(".")
import TaskCallNumParamEditor; sys.stdout.write(".")
import TaskEditCharScales; sys.stdout.write(".")
import TaskPicUpParamsEditor; sys.stdout.write(".")
import TaskExportToPicUp; sys.stdout.write(".")
import TaskCallPicUp3D; sys.stdout.write(".")
import TaskCallPicUp3DReload; sys.stdout.write(".")
import TaskProjectControler; sys.stdout.write(".")
import TaskProjectLoaderFormat2; sys.stdout.write(".")
```

```

from Bin.Tasks.TaskGeomManager      import TaskGeomManager; sys.stdout.write(".")
from Bin.Tasks.TaskExecuteCmdAsDemaon import TaskExecuteCmdAsDemaon; sys.stdout.write(".")
from Bin.Tasks.TaskAbout            import TaskAbout; sys.stdout.write(".")
print ""
class TasksList:
    '''

```

Define the dependence tree of all Tasks defined in the SPIS-UI framework.
If you wish add de new Tasks, you must:

1) Add it import in this module, as follow:

```
from Bin.Tasks.TaskMyTask      import TaskMyTask; sys.stdout.write(".")
```

2) Add it in the tasklist self.tasks, as follow:

```
(TaskMesher("iMyTask", 0, "Other")                                     , "MyTask")
```

The 0 flag means that this task must be run in foreground mode.

"Other" is the name of the Task on which one your task is dependent on.

The last field "MyTask" is the string corresponding the calling message used by the TaskManager to perform the task. Typically, this string is returned by the main SPIS-UI GUI.

Please see the SPIS-UI Developer Guide for further informations.

```

'''

def __init__(self):
    #print "Tasks list intantiation"
    self.isDeamon = 1
    self.tasks = []

def initTasksList(self):
    '''
    initialises the depedence tree (tasklist).
    '''
    # dependency tree between tasks.
    print "Building of the dependence tree..."
    #print "isDeamon", self.isDeamon
    self.tasks = [(TaskFileChooser("FileChooser")                                     , "FileChooser"),
                  (TaskCADImporter("CADImporter", 0)                             , "CADImporter"),
                  (TaskMesher("Mesher", 0, "CADImporter")                         , "Mesher"),
                  (TaskViewPipeline1("ViewPipeline1", 0, "ConvertGroup")          , "ViewPipeline1"),
                  (TaskMaterial("Material")                                       , "Material"),
                  (TaskGroupManager("GroupManager", 0, "Material")                , "GroupManager"),
                  (TaskFieldManager("FieldManager", 0, "ConvertGroup", \
                  "InitFields")                                     , "FieldManager"),
                  (TaskSpisNumInterface("SpisNumInterface", 0, \
                  "FieldManager", \
                  "ParamsEditor"), "SpisNumInterface"),
                  (TaskViewPipeline2("ViewPipeline2", self.isDeamon)              , "ViewPipeline2"),
                  (TaskJyTop("JyTop", self.isDeamon)                            , "JyTop"),
                  (TaskToolCaller("ToolCaller", self.isDeamon)                   , "ToolCaller"),
                  (TaskEditIni("EditIni", self.isDeamon)                        , "EditIni"),
                  (TaskPrompt("Prompt", self.isDeamon)                          , "Prompt"),
                  (TaskMesher2D("Mesher2D", 0, "CADImporter")                    , "Mesher2D"),
                  (TaskMesher3D("Mesher3D", 0, "CADImporter")                    , "Mesher3D"),
                  (TaskBuildDataFieldChooser("BuildDataFieldChooser", self.isDeamon) ,
"BuildDataFieldChooser"),
                  (TaskBuildPlot2D("BuildPlot2D",self.isDeamon)                  , "BuildPlot2D"),
                  (TaskGroupEditor("GroupEditor", 0)                            , "GroupEditor"),
                  (TaskReloadSolver("ReloadSolver")                              , "ReloadSolver"),
                  (TaskSaveProj("SaveProj", self.isDeamon)                      , "SaveProj"),
                  (TaskSaveProjAs("SaveProjAs",self.isDeamon)                    , "SaveProjAs"),
                  (TaskLoadProj("LoadProj")                                      , "LoadProj"),
                  (TaskMaterialEditor("MaterialEditor", self.isDeamon)            , "MaterialEditor"),

```

```

(TaskElecNodeEditor("ElecNodeEditor", self.isDeamon)      , "ElecNodeEditor"),
(TaskPlasmaEditor("PlasmaEditor", self.isDeamon)         , "PlasmaEditor"),
(TaskDataEditor("DataEditor", self.isDeamon)             , "DataEditor"),
(TaskReload("Reload")                                    , "Reload"),
(TaskInitGroup("InitGroup")                              , "InitGroup"),
(TaskConvertGroup("ConvertGroup", 0, "Mesher3D")          , "ConvertGroup"),
(TaskSaveCAD("SaveCAD", self.isDeamon)                   , "SaveCAD"),
(TaskLoadCAD("LoadCAD")                                  , "LoadCAD"),
(TaskSaveProperties("SaveProperties", self.isDeamon)       , "SaveProperties"),
(TaskLoadProperties("LoadProperties")                      , "LoadProperties"),
(TaskNewProject("NewProject")                            , "NewProject"),
(TaskSolverInit("SolverInit", 0)                         , "SolverInit"),
(TaskSolverBuildSim("SolverBuildSim")                    , "SolverBuildSim"),
(TaskSolverRun("SolverRun", self.isDeamon)               , "SolverRun"),
(TaskSolverReadSim("SolverReadSim")                      , "SolverReadSim"),
(TaskParaview("Paraview", self.isDeamon)                 , "Paraview"),
(TaskParamsEditor("ParamsEditor", 0)                    , "ParamsEditor"),
(TaskSaveGroups("SaveGroups")                            , "SaveGroups"),
(TaskLoadGroups("LoadGroups")                           , "LoadGroups"),
(TaskDocCaller("DocCaller", self.isDeamon)               , "DocCaller"),
(TaskCallCassandra("CallCassandra", self.isDeamon)       , "CallCassandra"),
(TaskCallJyConsole("CallJyConsole", self.isDeamon)      , "CallJyConsole"),
(TaskCallMemoryMonitor("CallMemoryMonitor", self.isDeamon),
"CallMemoryMonitor"),
(TaskCallJSynoptic("CallJSynoptic", self.isDeamon)       , "CallJSynoptic"),
(TaskMeshSplitter("MeshSplitter", 0, "InitFields")       , "MeshSplitter"),
(TaskInitFields("InitFields", 0, "ConvertGroup")         , "InitFields"),
(TaskExit("Exit")                                        , "Exit"),
(TaskExportToGmshMesh("ExportToGmshMesh", self.isDeamon) , "ExportToGmshMesh"),
(TaskMeshImporter("MeshImporter")                        , "MeshImporter"),
(TaskCallNumParamEditor("CallNumParamEditor")            ,
"CallNumParamEditor"),
(TaskDataBusCleaner("DataBusCleaner")                    , "DataBusCleaner"),
(TaskEditCharScales("EditCharScales")                    , "EditCharScales"),
(TaskPicUpParamsEditor("PicUpParamsEditor")              , "PicUpParamsEditor"),
(TaskExportToPicUp("ExportToPicUp" 0, "EditCharScales", "PicUpParamsEditor"),
"ExportToPicUp"),
(TaskCallPicUp3D("CallPicUp3D", 0, "PicUpParamsEditor", "ExportToPicUp"),
"CallPicUp3D"),
(TaskCallPicUp3DReload("CallPicUp3DReload"), "CallPicUp3DReload"),
(TaskProjectControler("ProjectControler"), "ProjectControler"),
(TaskProjectLoaderFormat2("ProjectLoaderFormat2", self.isDeamon),
"ProjectLoaderFormat2"),
(TaskGeomManager("GeomManager", self.isDeamon), "GeomManager"),
(TaskExecuteCmdAsDemaon("ExecuteCmdAsDemaon", self.isDeamon),
"ExecuteCmdAsDemaon"),
(TaskAbout("About", self.isDeamon), "About")
]

def setAllAlive(self):
    '''
    Forces all tasks to be in foreground mode. Usefull in script or batch mode.
    '''
    self.isDeamon = 0

```

This page is left intentionally blank.

10 Annexe 5: Example of NASCAP material catalog

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Document   : NascapMaterials.xml
Created on  : August 24, 2009, 3:23 PM
Author      : seb
Description:
    List nascap materials properties
-->

<Assembly>
  <MaterialProperties Name="Aluminum" Color="16776960" xmlns="material_schema">
    <Property Index="0" Value="1.0" />
    <Property Index="1" Value="0.0010" />
    <Property Index="2" Value="-1.0" />
    <Property Index="3" Value="13.0" />
    <Property Index="4" Value="0.97" />
    <Property Index="5" Value="0.3" />
    <Property Index="6" Value="154.0" />
    <Property Index="7" Value="0.8" />
    <Property Index="8" Value="220.0" />
    <Property Index="9" Value="1.76" />
    <Property Index="10" Value="0.244" />
    <Property Index="11" Value="230.0" />
    <Property Index="12" Value="4.0E-5" />
    <Property Index="13" Value="-1.0" />
    <Property Index="14" Value="26.98" />
    <Property Index="15" Value="2699.0" />
    <Property Index="16" Value="17.0" />
    <Property Index="17" Value="18.0" />
    <Property Index="18" Value="1.0E-16" />
    <Property Index="19" Value="20.0" />
  </MaterialProperties>
  <MaterialProperties Name="Gold" Color="65535" xmlns="material_schema">
    <Property Index="0" Value="1.0" />
    <Property Index="1" Value="0.0010" />
    <Property Index="2" Value="-1.0" />
    <Property Index="3" Value="79.0" />
    <Property Index="4" Value="0.88" />
    <Property Index="5" Value="0.8" />
    <Property Index="6" Value="88.8" />
    <Property Index="7" Value="0.92" />
    <Property Index="8" Value="53.5" />
    <Property Index="9" Value="1.73" />
    <Property Index="10" Value="0.413" />
    <Property Index="11" Value="135.0" />
    <Property Index="12" Value="2.9E-5" />
    <Property Index="13" Value="-1.0" />
    <Property Index="14" Value="197.0" />
    <Property Index="15" Value="19320.0" />
    <Property Index="16" Value="17.0" />
    <Property Index="17" Value="18.0" />
    <Property Index="18" Value="19.0" />
    <Property Index="19" Value="20.0" />
  </MaterialProperties>
  <MaterialProperties Name="Kapton" Color="32896" xmlns="material_schema">
    <Property Index="0" Value="3.5" />
    <Property Index="1" Value="1.27E-4" />
    <Property Index="2" Value="1.0E-16" />
  </MaterialProperties>
</Assembly>
```



```

<Property Index="3" Value="5.0" />
<Property Index="4" Value="2.1" />
<Property Index="5" Value="0.15" />
<Property Index="6" Value="71.48" />
<Property Index="7" Value="0.6" />
<Property Index="8" Value="312.1" />
<Property Index="9" Value="1.77" />
<Property Index="10" Value="0.455" />
<Property Index="11" Value="140.0" />
<Property Index="12" Value="2.0E-5" />
<Property Index="13" Value="1.0E16" />
<Property Index="14" Value="12.01" />
<Property Index="15" Value="1600.0" />
<Property Index="16" Value="17.0" />
<Property Index="17" Value="18.0" />
<Property Index="18" Value="1.0E-18" />
<Property Index="19" Value="20.0" />
</MaterialProperties>
<MaterialProperties Name="Teflon" Color="255" xmlns="material_schema">
  <Property Index="0" Value="2.0" />
  <Property Index="1" Value="1.27E-4" />
  <Property Index="2" Value="1.0E-16" />
  <Property Index="3" Value="7.0" />
  <Property Index="4" Value="3.0" />
  <Property Index="5" Value="0.3" />
  <Property Index="6" Value="45.4" />
  <Property Index="7" Value="0.4" />
  <Property Index="8" Value="218.0" />
  <Property Index="9" Value="1.77" />
  <Property Index="10" Value="0.455" />
  <Property Index="11" Value="140.0" />
  <Property Index="12" Value="2.0E-5" />
  <Property Index="13" Value="1.0E16" />
  <Property Index="14" Value="19.0" />
  <Property Index="15" Value="2150.0" />
  <Property Index="16" Value="17.0" />
  <Property Index="17" Value="18.0" />
  <Property Index="18" Value="1.0E-16" />
  <Property Index="19" Value="20.0" />
</MaterialProperties>
<MaterialProperties Name="OSR" Color="65280" xmlns="material_schema">
  <Property Index="0" Value="4.8" />
  <Property Index="1" Value="1.5E-4" />
  <Property Index="2" Value="1.0E-17" />
  <Property Index="3" Value="10.0" />
  <Property Index="4" Value="3.3" />
  <Property Index="5" Value="0.5" />
  <Property Index="6" Value="116.3" />
  <Property Index="7" Value="0.81" />
  <Property Index="8" Value="183.1" />
  <Property Index="9" Value="1.86" />
  <Property Index="10" Value="0.455" />
  <Property Index="11" Value="140.0" />
  <Property Index="12" Value="2.0E-5" />
  <Property Index="13" Value="1.0E19" />
  <Property Index="14" Value="20.0" />
  <Property Index="15" Value="2660.0" />
  <Property Index="16" Value="17.0" />
  <Property Index="17" Value="18.0" />
  <Property Index="18" Value="1.0E-18" />
  <Property Index="19" Value="20.0" />

```

```

</MaterialProperties>
<MaterialProperties Name="Black Kapton" Color="32768" xmlns="material_schema">
  <Property Index="0" Value="3.5" />
  <Property Index="1" Value="2.5E-6" />
  <Property Index="2" Value="-1.0" />
  <Property Index="3" Value="5.0" />
  <Property Index="4" Value="2.1" />
  <Property Index="5" Value="0.15" />
  <Property Index="6" Value="71.48" />
  <Property Index="7" Value="0.6" />
  <Property Index="8" Value="312.1" />
  <Property Index="9" Value="1.77" />
  <Property Index="10" Value="0.455" />
  <Property Index="11" Value="140.0" />
  <Property Index="12" Value="5.0E-6" />
  <Property Index="13" Value="-1.0" />
  <Property Index="14" Value="12.01" />
  <Property Index="15" Value="1600.0" />
  <Property Index="16" Value="17.0" />
  <Property Index="17" Value="18.0" />
  <Property Index="18" Value="1.0E-16" />
  <Property Index="19" Value="20.0" />
</MaterialProperties>
<MaterialProperties Name="Solar Cells" Color="16724787" xmlns="material_schema">
  <Property Index="0" Value="3.8" />
  <Property Index="1" Value="1.25E-4" />
  <Property Index="2" Value="1.0E-17" />
  <Property Index="3" Value="10.0" />
  <Property Index="4" Value="5.8" />
  <Property Index="5" Value="1.0" />
  <Property Index="6" Value="77.5" />
  <Property Index="7" Value="0.45" />
  <Property Index="8" Value="156.1" />
  <Property Index="9" Value="1.73" />
  <Property Index="10" Value="0.244" />
  <Property Index="11" Value="230.0" />
  <Property Index="12" Value="2.0E-5" />
  <Property Index="13" Value="1.0E19" />
  <Property Index="14" Value="20.0" />
  <Property Index="15" Value="2660.0" />
  <Property Index="16" Value="17.0" />
  <Property Index="17" Value="18.0" />
  <Property Index="18" Value="1.0E-18" />
  <Property Index="19" Value="20.0" />
</MaterialProperties>
<MaterialProperties Name="Npaint" Color="16777215" xmlns="material_schema">
  <Property Index="0" Value="3.5" />
  <Property Index="1" Value="5.0E-5" />
  <Property Index="2" Value="5.9E-14" />
  <Property Index="3" Value="5.0" />
  <Property Index="4" Value="2.1" />
  <Property Index="5" Value="0.15" />
  <Property Index="6" Value="-1.0" />
  <Property Index="7" Value="0.0" />
  <Property Index="8" Value="1.05" />
  <Property Index="9" Value="9.8" />
  <Property Index="10" Value="0.455" />
  <Property Index="11" Value="140.0" />
  <Property Index="12" Value="2.0E-5" />
  <Property Index="13" Value="1.0E13" />
  <Property Index="14" Value="12.01" />

```

```
<Property Index="15" Value="1600.0" />
<Property Index="16" Value="17.0" />
<Property Index="17" Value="18.0" />
<Property Index="18" Value="1.0E-18" />
<Property Index="19" Value="20.0" />
</MaterialProperties>
<MaterialProperties Name="Graphite" Color="8421504" xmlns="material_schema">
  <Property Index="0" Value="3.5" />
  <Property Index="1" Value="1.27E-4" />
  <Property Index="2" Value="-1.0" />
  <Property Index="3" Value="6.0" />
  <Property Index="4" Value="1.0" />
  <Property Index="5" Value="0.3" />
  <Property Index="6" Value="-1.0" />
  <Property Index="7" Value="0.0" />
  <Property Index="8" Value="2.0" />
  <Property Index="9" Value="12.0" />
  <Property Index="10" Value="0.455" />
  <Property Index="11" Value="140.0" />
  <Property Index="12" Value="7.2E-6" />
  <Property Index="13" Value="-1.0" />
  <Property Index="14" Value="12.01" />
  <Property Index="15" Value="1600.0" />
  <Property Index="16" Value="17.0" />
  <Property Index="17" Value="18.0" />
  <Property Index="18" Value="1.0E-18" />
  <Property Index="19" Value="20.0" />
</MaterialProperties>
</Assembly>
```

11 Annexe 6: Example of extended material

Tabulated fields are not fully supported in the current version of SPIS.

```
<MaterialProperties Name="Aluminum" Color="16776960" xmlns="material_schema">
  <Property Index="0" Value="1.0" />
  <Property Index="1" Value="0.0010" />
  <Property Index="2" Value="-1.0" />
  <Property Index="3" Value="13.0" />
  <Property Index="4" Value="0.97" />
  <Property Index="5" Value="0.3" />
  <Property Index="6" Value="154.0" />
  <Property Index="7" Value="0.8" />
  <Property Index="8" Value="220.0" />
  <Property Index="9" Value="1.76" />
  <Property Index="10" Value="0.244" />
  <Property Index="11" Value="230.0" />
  <Property Index="12" Value="4.0E-5" />
  <Property Index="13" Value="-1.0" />
  <Property Index="14" Value="26.98" />
  <Property Index="15" Value="2699.0" />
  <Property Index="16" Value="17.0" />
  <Property Index="17" Value="18.0" />
  <Property Index="18" Value="1.0E-16" />
  <Property Index="19" Value="20.0" />
  <ExtendedSpisProperty Name="Prop1"
    Description = "This is an example of extended property"
    Type="float" Unit="m.s^-1" Value="300" />
  <ExtendedSpisProperty Name="Prop2" Type="int" Unit="None" Value="1" />
  <ExtendedSpisProperty Name="Prop3"
    TypesTable="float, float"
    UnitsTable="eV, kg.s^-1"
    ValuesTable="1.0, 2.45,
                2.0, 2.46,
                3.0, 3.00,
                4.0, 3.5,
                5.0, 4.0"
  />
  <ExtendedSpisProperty Name="Prop4"
    TypesTable="float, float, float"
    UnitsTable="eV, rad, kg.s^-1"
    ValuesTable="1.0, 0, 2.45,
                2.0, 1.0471975511965976, 2.46,
                3.0, 1.5707963267948966, 3.00,
                4.0, 3.14159, 3.5,
                5.0, 6.3, 4.0"
  />
  <ExtendedSpisProperty Name="Prop5"
    Type="File" Unit="None" Value="./myDataFile.dat" />
  <ExtendedSpisProperty Name="Comment1"
    Type="String"
    Unit="None"
    Value="Just to comment, this is a beautifull extended
          NASCAP/SPIS material." />
</MaterialProperties>
```